uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Electrónica Industrial y Automatización
2017-2018

Trabajo de Fin de Grado

"Análisis de registros de EEG mediante redes neuronales"

Anaëlle Gordillo Dagallier

Tutor

Enrique Pelayo Campillos

Lugar y fecha de presentación prevista

10 de octubre de 2018, 9:30 en el aula 1.2.C.16 de la EPS UC3M

RESUMEN

Uno de los objetivos de la neurociencia actual es la comprensión de los mecanismos por los que los seres humanos realizan la clasificación y el reconocimiento de objetos. En los últimos años, se ha buscado determinar las áreas del cerebro que se encargan de dicho proceso. Se han realizado numerosos estudios con este fin, partiendo de señales procedentes de magneto- y electroencefalogramas [3][11], a las que se les aplican algoritmos basados en el análisis de patrones multivariantes.

En este proyecto de fin de grado se parte de la investigación conducida por Kaneshiro et al. [21] en la que, usando señales electroencefalográficas procedentes de diez sujetos de características heterogéneas, aplicaron un análisis de similitud representacional (RSA) para entender el procesamiento de objetos y su clasificación en seis categorías (caras humanas, partes del cuerpo humano, caras de animales, partes de cuerpos animales, frutas/vegetales, y objetos inanimados). Este TFG propone aplicar en su lugar, técnicas de aprendizaje profundo, en auge en la actualidad, a la misma base de datos para resolver el problema de clasificación planteado por Kaneshiro.

Se han desarrollado tres modelos básicos de redes neuronales, de los cuales dos son convolucionales (con una o dos capas de convolución). Posteriormente, se han implementado mejoras como la validación cruzada, tomando los modelos más sencillos como base. En concreto, con el método de validación cruzada basado en una estructura con dos capas convolucionales, aplicado al conjunto total de datos, se ha obtenido una precisión superior al 70% al predecir la categoría de nuevas imágenes procedentes de sujetos con los que se ha entrenado la red neuronal. Con esta técnica, se han conseguido mejorar ampliamente los resultados obtenidos en el estudio del que surge este proyecto, en el que la tasa de acierto alcanzada no supera el 45%.

Palabras clave
Clasificación de imágenes, aprendizaje profundo, redes convolucionales, electroencefalografía (EEG)

ABSTRACT

One of the goals of actual neuroscience is to understand the mechanisms by which we, humans, recognise and classify objects. In recent years, it has been tried to determine the areas of the brain dedicated to this process. For this, a number of studies have been conducted using as raw material signals coming from magneto- and electroencephalograms [3][11], and applying a multivariate pattern analysis to them.

In this bachelor's thesis, the investigation performed by Kaneshiro et Al. [21] is used as reference. In their study, starting from EEG signals coming from 10 subjects belonging to a heterogeneous group, they applied a Representational Similarity Analysis (RSA) to comprehend the dynamics of object processing and their classification in one out of six possible categories: human face, human body, animal face, animal body, fruit/vegetable, and inanimate objects. This project proposes to change the approach to the classification problem raised by Kaneshiro, implementing on the same dataset deep learning algorithms, which are experimenting great improvements nowadays.

Three basic artificial neural networks have been developed, being two of them of the convolutional kind. Some optimization methods have been proposed, like a cross-validation method that uses the simpler networks as its core. In fact, a cross-validation algorithm based on a convolutional network with two convolutional layers, applied over the complete dataset, has shown an accuracy greater than 70% when the category of new data from one of the subjects of the training set is predicted. With this implementation, the results of the original paper have been greatly improved, since their achieved accuracy was lower than a 45%.

Keywords

Image classification, deep learning, convolutional neural networks, electroencephalography (EEG)

DEDICATORIA

Quisiera darle las gracias a todas las personas que han estado apoyándome y enseñándome a avanzar poco a poco a lo largo de esta carrera. En concreto,

A ‘Un mundo feliz’, por hacer de estos años toda una aventura llena de risas y maratones. Sin vosotros no habría sido lo mismo.

A Alba, porque, aunque hayan pasado meses sin que nos veamos, el tiempo para nosotras no pasa. Las tonterías y sonrisas, y todas las charlas de ánimo que me has dado, han hecho que esté hoy aquí.

A mis amigos del Erasmus, por abrirme la mirada a nuevos mundos y hacerme descubrir las cosas desde otra perspectiva.

A mi padre, por enseñarme desde pequeña que no hay que abandonar y siempre merece la pena seguir luchando por lo que se quiere.

A todos los profesores que han sabido transmitirnos su pasión por lo que hacen, animándonos a continuar esforzándonos por mejorar. Y, sobre todo, a mi tutor, Enrique, por tu infinita paciencia a lo largo de estos últimos meses y por implicarte tanto en este proyecto.

A mi madre, porque siempre me has ayudado en lo que podías y me has levantado cada mañana con una sonrisa y ánimos renovados para avanzar y no estancarme en mi cabezonería.

A Lore, porque siempre has sido mi fuente de inspiración y apoyo. Gracias por todas esas noches de conversaciones y por mostrarme el tipo de persona que quiero llegar a ser.

A Jorge, simplemente por estar siempre ahí, aunque haya medio mundo de por medio. Gracias por todo.

Y a Salva, por enseñarme todo lo que se puede conseguir con buena voluntad y por darme esa motivación por la que seguir luchando siempre.

Gracias a todos por hacerme quien soy.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación

El reconocimiento de objetos es clave para la supervivencia de los seres humanos en aspectos como la búsqueda de alimento, la elusión de peligros, el cuidado de la descendencia, la elección de la pareja sexual, la orientación y las interacciones sociales. Se consagra como uno de los factores más importantes de la percepción visual, siendo, a su vez, el menos entendido.

Desde hace años, el hombre busca entender el proceso mediante el cual el cerebro interpreta los estímulos visuales que recibe y los clasifica por categorías o especies. Empezando por estudios con animales, especialmente con monos como los macacos [1], se ha buscado determinar las áreas del cerebro que se encargan de este proceso de clasificación y reconocimiento [2]. Se ha conseguido determinar mediante métodos no invasivos que, en el lóbulo occipital complejo del cerebro, se genera una respuesta ante la percepción de un estímulo visual. En [3] se ha comprobado que dicha actividad no se relaciona con posibles fuentes de ruido, como podrían serlo la frecuencia espacial de Fourier presente en las imágenes, los movimientos del ojo o cambios en el tipo de filtrado o en las dimensiones de los estímulos. Además, usando imaginería de resonancia magnética funcional (fMRI), se ha detectado que un área del giro fusiforme se activa de forma significativa al reconocer caras [4] (en el Anexo 1 se muestran las distintas partes del cerebro). Así mismo, se ha establecido en otros experimentos, como [5][6], que distintas regiones corticales del córtex ventral temporal se relacionan con el reconocimiento de tipos de objetos concretos, como edificios o letras. Sin embargo, son numerosos los estudios que contradicen esta estipulación, determinando que el córtex cerebral no se compone de un mosaico de módulos específicos a cada categoría de objeto, sino que se trata de un medio continuo en el que se forman combinaciones de áreas en un orden topológico concreto [7][8].

1.2. Objetivos

En este proyecto se busca como objetivo principal relacionar las imágenes que ve un sujeto, con la respuesta que se genera en su cerebro al reconocer la fotografía o dibujo. Por tanto, se trata de clasificar lecturas procedentes de un electroencefalograma en una serie de categorías establecidas.

Para cumplir este objetivo, se proponen diversos modelos de redes neuronales profundas: simples y convolucionales.

Para establecer si se alcanzan los objetivos, se comprueba la precisión y error de los modelos en tres supuestos:

- La introducción de un dato nuevo del mismo sujeto con el que se ha realizado el entrenamiento de la red.
- La introducción de un dato de un sujeto diferente a los sujetos con los que se ha entrenado la red.

- La introducción de un dato nuevo de uno de los sujetos con los que se ha entrenado la red.

1.3. Estado del arte

En este apartado se recogen otros métodos e investigaciones que buscan. Existen estudios en los que se obtienen las lecturas a través de resonancias magnéticas

En este apartado se presentan las técnicas y estudios actuales que tratan de entender el funcionamiento del cerebro humano a la hora de reconocer estímulos visuales. De esta forma, aparecen investigaciones que se basan en dos tipos de señales: EEG, obtenidas mediante una electroencefalografía; y MEG, procedentes de una magnetoencefalografía. Ambos métodos comparten la misma base y cuentan con resoluciones espaciales similares, pero presentan sensibilidades diferentes en función de la orientación y la profundidad de los tejidos [9]. Estudios recientes han investigado la forma en la que se clasifican las categorías, empleando el método de clasificación de patrones multivariados. Esta técnica permite analizar las señales EEG o MEG en su totalidad, sin tener que preseleccionar componentes espaciales o temporales [10][11]. Diversos estudios han demostrado que, mediante este tipo de clasificación, se pueden distinguir hasta seis categorías diferentes [12][13][14][15] e incluso dibujos hechos a base de líneas, y nombres de objetos escritos y hablados [11][16]. También hay un estudio que ha tomado el enfoque contrario, prediciendo las activaciones de tipo MEG a partir de características concretas de estímulos visuales [17].

Los resultados de estas investigaciones han llevado a realizar un estudio de la estructura representacional de las categorías de los objetos. Esta estructura se puede analizar mediante el Análisis de Similitudes Representacionales (RSA) [18], consistente en comparar las categorías de las respuestas cerebrales no en función de las respuestas en sí, sino a partir de las diferencias distales existentes entre ellas, como queda recogido en una matriz de disimilitud representacional (RDM). En algunos estudios, se han empleado técnicas de disimilitud representacional basados en pares de correlaciones [19], o pares de precisión de clasificadores [20]. En otros, como la investigación de Kaneshiro et Al. [21], construyen la matriz RDM a partir de las matrices de confusión obtenidas de una clasificación multiclase.

Con un enfoque completamente diferente, se han propuesto competiciones de aprendizaje profundo para resolver el problema en cuestión. Siguiendo esta línea, se han llevado a cabo hackathones de *Deep learning*, como el propuesto por la empresa Deepgram (deepgram.com) en 2017. Debido a este evento, surge la idea de llevar a cabo el trabajo de fin de grado desarrollado en esta memoria, y aplicar las redes neuronales artificiales al problema de correlacionar imágenes con las señales EEG generadas en el cerebro.

1.4. Electroencefalografía

La electroencefalografía o EEG, es el proceso mediante el cual se obtiene un registro de la actividad eléctrica del cerebro. Se trata de una técnica no invasiva, porque los electrodos se colocan sobre el cuero cabelludo. Las señales obtenidas mediante esta técnica reflejan las fluctuaciones en el potencial eléctrico del cerebro generado por un grupo de neuronas, no siendo posible obtener el voltaje relativo a una sola neurona sin recurrir a métodos invasivos.

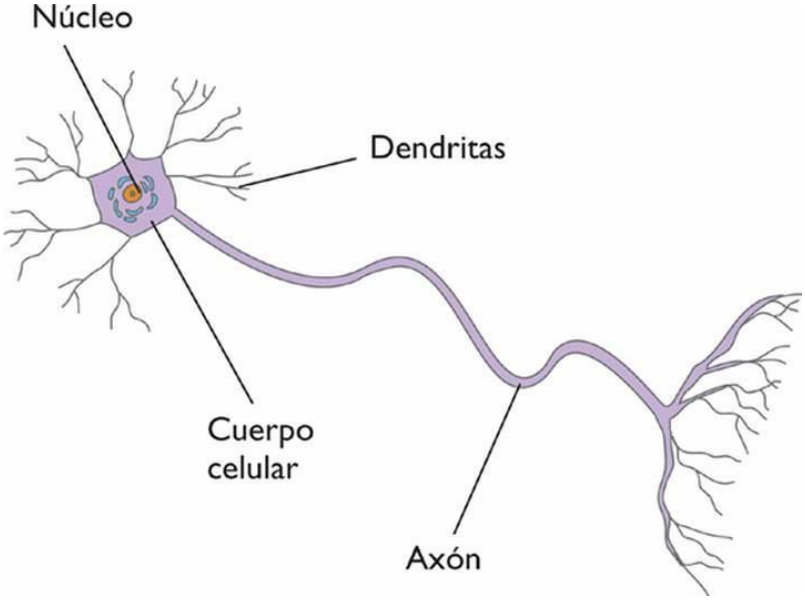


Fig. 1. Partes de una neurona [22]

La neurona es la unidad básica del sistema nervioso. Se compone de diferentes partes: dendritas, a través de las cuales recibe la información; núcleo, donde la procesa; y axón, mediante el cual transmite el impulso eléctrico a la siguiente neurona, fibra muscular o glándula. Estas transmisiones son de carácter iónico, por lo que para poder convertirlas a la corriente eléctrica que los electrodos transmiten al amplificador, se coloca un gel o solución salina entre el cuero cabelludo y el electrodo.
Finalmente, las señales captadas son amplificadas y representadas frente al tiempo, obteniendo ondas como las que se muestran en la figura 2.

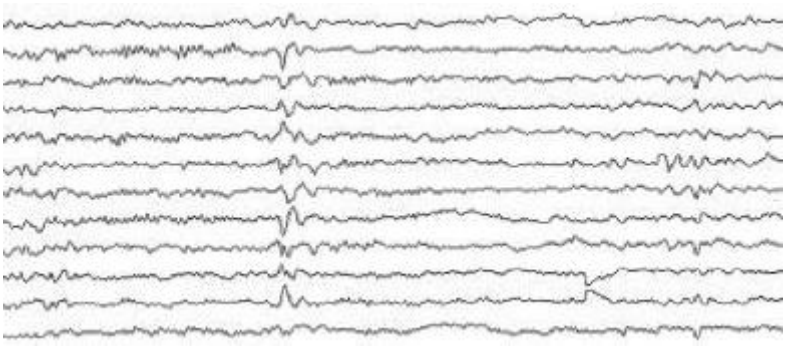


Fig. 2. Ejemplo de encefalograma.

Cuando se produce un estímulo externo, como la presentación de una imagen, el sistema nervioso sigue el siguiente proceso. Capta el estímulo a través del sistema visual, que es un exteroceptor. A continuación, retransmite este estímulo a través del nervio óptico hasta una componente integradora donde se analiza. Esta componente se compone de numerosas neuronas que comparten la información mediante impulsos eléctricos. Si el estímulo requiere una respuesta, esta se comunica a las fibras musculares mediante las neuronas motoras, o a las glándulas, generando una secreción glandular. Cada una de las respuestas que puede generar nuestro cuerpo, crea un potencial eléctrico de diferentes intensidades en distintas zonas del cerebro. Es el estudio de estas diferencias lo que nos permite determinar el tipo de respuesta que se está dando.

Los electrodos empleados en las electroencefalografías no discriminan las corrientes eléctricas que captan. Por tanto, debido a la baja amplitud de voltaje que presentan las señales EEG, entre 20 y 100 uV, es muy probable que, a la hora de adquirir este tipo de señal, se obtengan interferencias con distintas procedencias. Este ruido es lo que se conoce como artefactos.

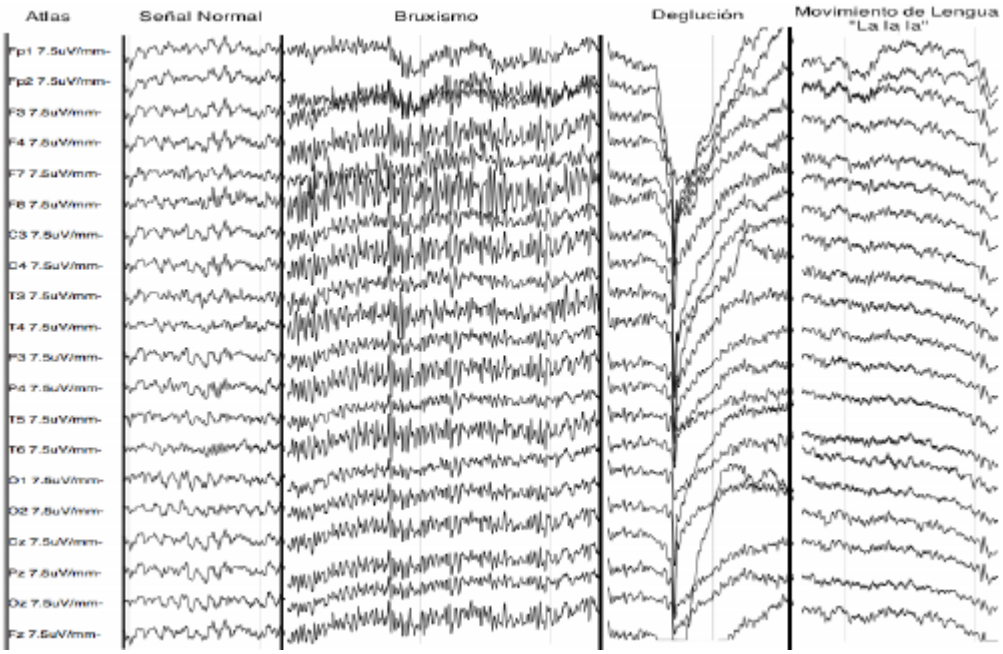


Fig. 3. Ejemplo de contaminación por artefactos de señal EEG [23].

Los artefactos pueden tener dos orígenes: fisiológico o extrafisiológico. Con los de origen fisiológico se hace referencia al ruido que genera el propio cuerpo humano. De esta forma, el movimiento ocular (EOG), el ritmo cardíaco (electrocardiograma, ECG) o cualquier movimiento de los músculos de la cara (electromiograma, EMG), pueden introducir ruido en la señal de interés. Su influencia puede reducirse mediante su análisis posterior, es decir, la interferencia de otros músculos, por ejemplo, genera una señal de ruido a una frecuencia determinada, con una duración y forma característicos, por lo que, si se aplica el filtro adecuado o un par de electrodos en posiciones no encefálicas, se puede eliminar su presencia de la señal EEG. Los globos oculares actúan como dipolos, por lo que su movimiento genera amplios picos de corriente alterna detectable por cualquiera de los electrodos. Existen varios métodos para reducir la contaminación producida por los ojos, como los que aparecen en [24]. En cuanto a los artefactos de la piel, son difícilmente suprimibles, debido a que existe un potencial de corriente continua considerable entre algunas capas de la piel y cualquier alteración o deformación local puede alterarlo. Se puede tratar de disminuir este artefacto limpiando la piel con alcohol, para reducir la resistencia que pueda existir.

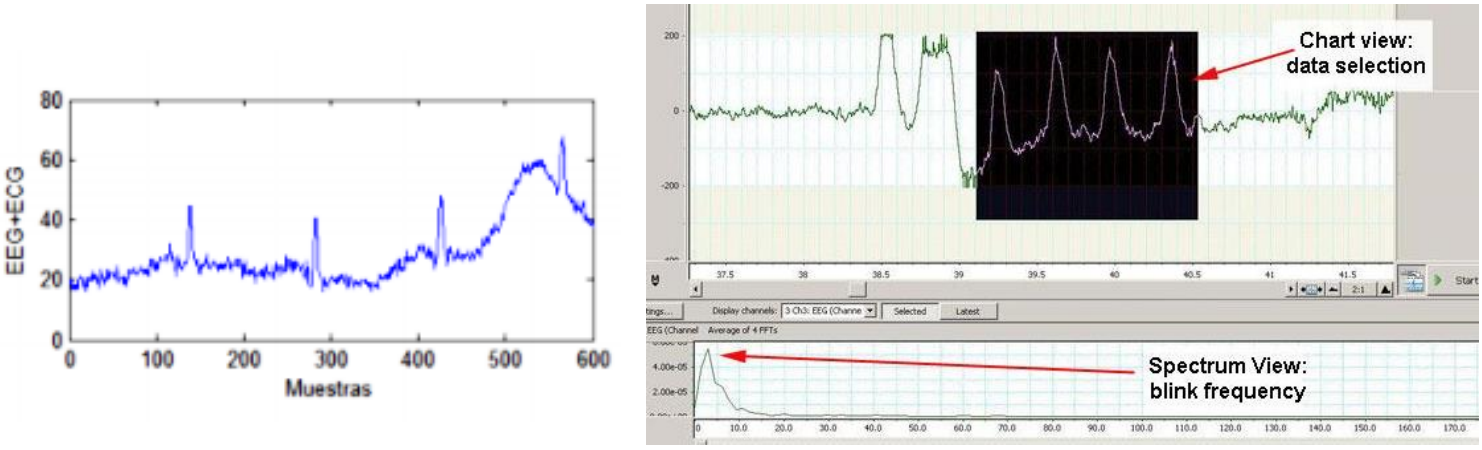


Fig. 4. Contaminación por artefacto cardíaco [25] (izquierda), y por parpadeo [26] (derecha).

Por otro lado, los artefactos extrafisiológicos pueden provocarse al tocar, y mover, los electrodos durante la toma de datos. Así mismo, si el contacto entre el electrodo y la piel no es el adecuado, actúa como una antena generando interferencias cíclicas a 60 Hz. Otra radiación de alta frecuencia procedente de otros aparatos electrónicos puede sobrecargar los amplificadores de EEG. Una forma de eliminar este tipo de artefacto es ir desconectando uno a uno todos los aparatos circundantes y repetir la medición, hasta dar con el causante de la interferencia.

1.5. Conceptos básicos de las redes neuronales

En este apartado se pretenden explicar todos los conceptos básicos necesarios para entender el proyecto que se ha desarrollado. Empezando por la base, como lo son las neuronas artificiales y su correspondencia con las neuronas físicas, y terminando por explicar el funcionamiento de las redes neuronales convolucionales, se explican las distintas funciones que se emplean a lo largo del proyecto.

1.5.1. Neurona artificial

El funcionamiento de una neurona artificial busca reproducir el funcionamiento de una neurona biológica. Por ello, para poder explicar el primer concepto, se ha de comparar con una versión muy simplificada del segundo [27].

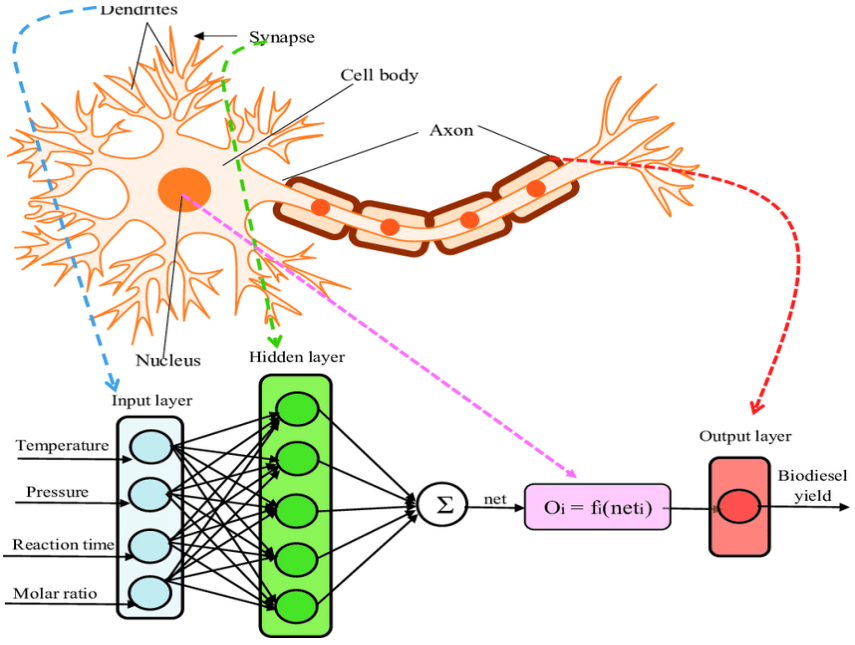


Fig. 5. Comparativa entre neurona biológica y neurona artificial [28].

Como se comentaba en el apartado anterior, la neurona es la unidad básica del sistema nervioso. Es de suponer entonces, que un nodo o neurona artificial será también la base de una red neuronal. La neurona recibe un estímulo a través de sus dendritas, y lo retransmite una vez procesado a otras neuronas a través de su axón. En el modelo matemático, la sinapsis que se produce entre el 'axón' de una neurona y las 'dendritas' de la siguiente, se visualiza como la multiplicación entre el vector de entrada *x* que transmite el axón, y la fuerza de la sinapsis con las dendritas, denominada matriz de pesos *w*. Estos pesos se pueden entrenar en el modelo y controlan la influencia de una neurona sobre otra. En el modelo biológico, en el núcleo de la neurona se suman todas las señales que recibe a través de sus dendritas y, si supera un cierto umbral, se transmiten impulsos eléctricos a través del axón hacia las siguientes neuronas. Ésto se modela como una función de activación *f* [29].

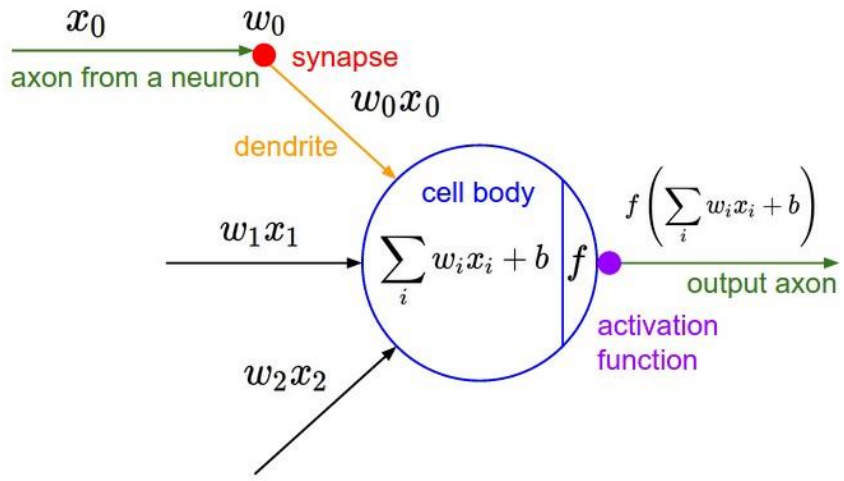


Fig. 6. Representación matemática de una neurona [29].

En el cuerpo de la neurona, tras realizar el sumatorio de todas las entradas, se añade un *bias* (*b*) para cambiar el rango de los pesos de entrada.

1.5.2. *Loss Function* o Función de pérdida

Esta función nos permite determinar el error con el que está respondiendo la red neuronal. Existen diversos tipos de funciones de pérdida.

El más habitual en problemas de clasificación binaria, es el *squared error loss*, o error cuadrático. Se define como la diferencia al cuadrado entre el valor real *y*, y el valor obtenido al pasar la entrada $\bar{x}$ por una neurona.

$$E(\overline{w}) = \left(y - \sigma\left(w_0 + \sum_{i=0}^{n} w_i x_i \right) \right)^2$$

Los parámetros que aparecen en la ecuación anterior son: el vector de pesos ($\overline{w}$), la salida real (y), la función de activación (σ), y la salida obtenida en la neurona i (x_i).

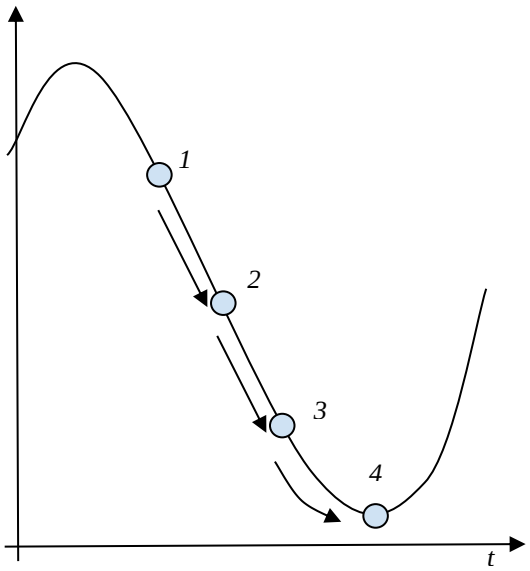
Sin embargo, para tareas de distribuciones de probabilidad, es preferible utilizar la función de pérdida de entropía cruzada (*cross-entropy loss*), puesto que solo pretende

maximizar la confianza del modelo hacia la clase correcta, sin atender a las distribuciones de probabilidad de las otras categorías. Para una clasificación de *k* clases, un vector concreto $\bar{y}$, comparado con su codificación *one-hot* $\vec{\hat{y}}$, la función de entropía cruzada es:

$$\mathcal{L}(\vec{y}, \vec{\hat{y}}) = -\sum_{i=1}^{k} \hat{y}_i \ln y_i$$

Como se observa en la ecuación anterior, la salida real de las neuronas que se predice que van a dar como *output* una clase incorrecta, es eliminada de la función de pérdida.

El objetivo es, por tanto, reducirla al máximo. Para ello existen diversos tipos de algoritmos de optimización que buscan reducir este error. El más común es el conocido como método de descenso por gradiente, o *gradient descent algorithm.* Dicho método consiste en minimizar la función de pérdida, actualizando el vector de pesos en la dirección de máxima pendiente de bajada de dicha función, escalado por el ratio de aprendizaje η(*learning rate*).



$$\bar{w} \leftarrow \bar{w} - \eta \frac{\partial E(\bar{w})}{\partial \bar{w}}$$

Fig. 7. Ejemplo del método de descenso por gradiente para un caso sencillo en 1D.

1.5.3. *Back and forward propagation*

En una red neuronal, el flujo de información es bidireccional: de la entrada hasta el resultado final, y del final al principio nuevamente en forma de error para mejorar la predicción que hace la red. Estos procesos se conocen como *forward* y *back propagation*.

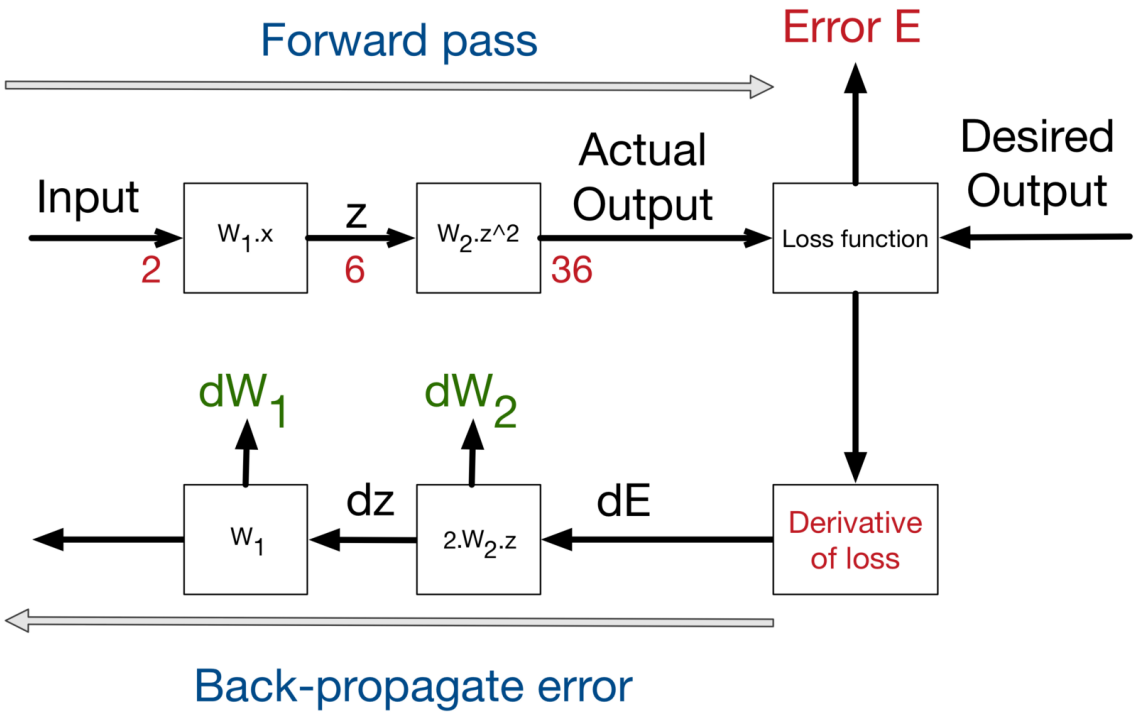


Fig. 8. Ejemplo de propagación y retropropagación 42[30].

En la propagación inicial, el dato o *input* pasa por las diferentes capas de las que se compone el modelo y en cada una de ellas se le aplica una transformación lineal o no lineal.

Durante la retropropagación, se compara la salida obtenida con la esperada, y el error que se obtiene, se propaga de forma inversa a través de la red (mediante la regla de la cadena) para actualizar el vector de pesos y *bias* y obtener una predicción que se aproxime más al valor real.

Lo que hace posible la retropropagación, son los gradientes o derivadas de las funciones de activación. Aquí entra en juego la regla de la cadena, para calcular la variabilidad del *output* (*y*) respecto al *input* (*x*).

$$\frac{\partial \mathrm{y}}{\partial \mathrm{x}} = \frac{\partial y}{\partial f} \cdot \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$

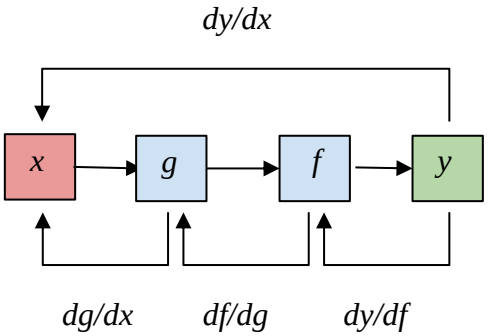


Fig. 9. Regla de la cadena.

1.5.4. Función de activación

Como se ha explicado en secciones anteriores, la neurona artificial aplica una transformación lineal sobre la información que recibe. Antes de transmitir esta señal

transformada a la siguiente neurona, es necesario establecer si la información es relevante o no. Esto se consigue mediante las funciones de activación. Además, gracias a estas funciones, es posible el *back-propagation*, ya que los gradientes se propagan junto a los errores para poder actualizar los pesos y bias, y sin la característica de derivabilidad de las transformaciones no lineales, esto no sería posible.

A continuación, se exponen los tipos de función de activación más conocidos:

- *Binary step function.* Ideal para problemas en los que se debe decidir si un elemento pertenece o no a una clase. Si el valor de la neurona es mayor que un umbral, la neurona se activa. En caso contrario, su respuesta se anula. Debido a la simplicidad de funcionamiento de esta función, no se usa habitualmente. Además, su derivada es igual a cero, por lo que el proceso de *back-propagation* no se puede llevar a cabo.

$$f(x) = \begin{cases} 1, if\ x \geq 0 \\ 0, if\ x < 0 \end{cases}$$

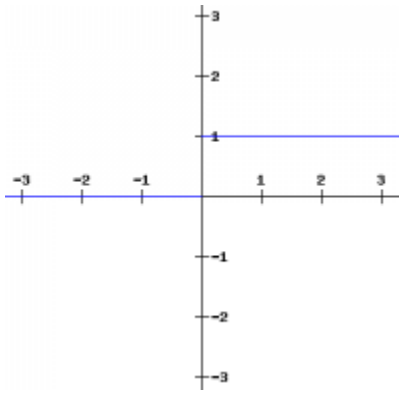


Fig. 10. Función de activación: *Binary step function.*

- Función lineal. Ejecuta otra transformación lineal sobre la señal, escalándola. Soluciona el problema que aparecía en el caso anterior, ya que su gradiente es distinto de cero. Sin embargo, es una constante, por lo que el error es independiente de la entrada que recibe la neurona. Por tanto, aunque se quiera hacer una red más profunda, la salida de la red sería una mera transformación lineal de la entrada, limitando el uso del modelo a tareas sencillas.

$$f(x) = 2x$$

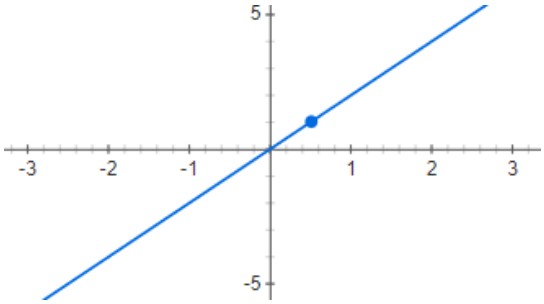


Fig. 11. Función de activación: Lineal.

- *Sigmoid*. Esta función con forma de S presenta como ventaja frente a las anteriormente presentadas que es no lineal, y, por tanto, derivable y su gradiente depende de la entrada. Gracias a esto, la salida de una red neuronal con varias neuronas activadas con esta función, es también no lineal y el vector de pesos puede ser correctamente actualizado mediante *back-propagation.* En

contrapartida, un nodo activado de esta manera tan solo transmite valores positivos, lo cual, dependiendo del propósito, puede no ser deseable. Además, fuera del intervalo (-3,3), el gradiente de la función tiende a cero, significando que el modelo aprende muy lentamente fuera de dicho intervalo.

$$f(x) = \frac{1}{1+e^{-x}}$$

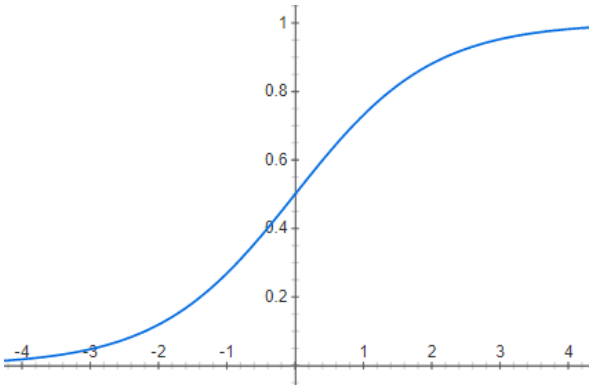


Fig. 12. Función de activación: *Sigmoid*

- Tangente hiperbólica (tanh). Se trata de una mejora de la función sigmoide puesto que puede ofrecer tanto valores positivos como negativos a la neurona siguiente. En los demás aspectos, se comporta igual que la mencionada función.

$$f(x) = \frac{2}{1+e^{-2x}} - 1$$

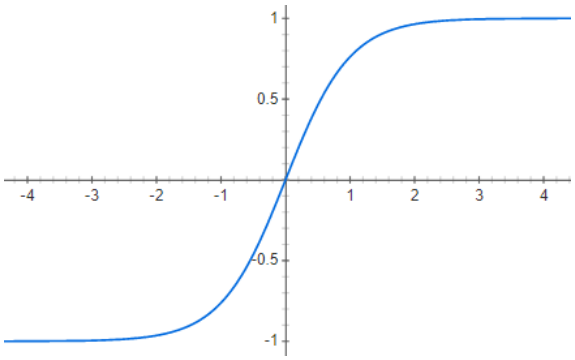


Fig. 13. Función de activación: Tangente hiperbólica.

- Unidad lineal rectificada o ReLU. La más usada actualmente en la solución de problemas de aprendizaje profundo. Esta función no lineal, permite que no se activen todas las neuronas al mismo tiempo, ya que aquellas con valores negativos, quedan anuladas. De esta forma, se promueve la velocidad y sencillez del cómputo de la red. Además, acelera la convergencia del *gradient descent* comparado con las funciones *sigmoid* y tanh (por un factor de 6 en [31]). En cambio, esta reducción a cero de los valores negativos resulta en la no-actualización de dichas neuronas (gradiente = 0), y puede generar neuronas que nunca se pueden activar.

$$f(x) = \max(0, x)$$

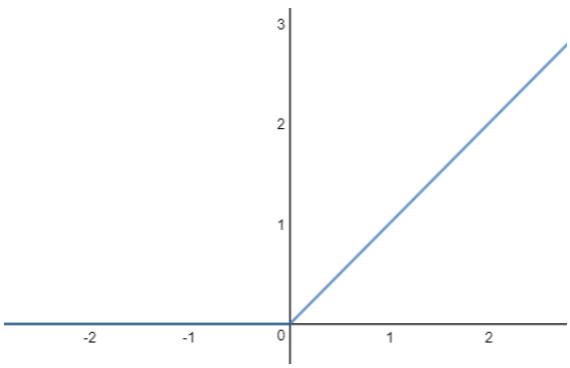


Fig. 14. Función de activación: ReLU.

Esta es la función de activación que se aplicará a todas las capas ocultas de los modelos propuestos en este proyecto.

- *Leaky ReLU.* Es la versión mejorada de la función ReLU. En lugar de anular las neuronas con valores negativos, les aplica una transformación lineal muy pequeña, solucionando así el problema de las neuronas 'muertas'. Además, si la pendiente de esta transformación lineal es entrenable, la función de activación se conoce como *parameterized ReLU*.

$$f(x) = \begin{cases} x, if\ x \geq 0 \\ ax, if\ x < 0 \end{cases}$$

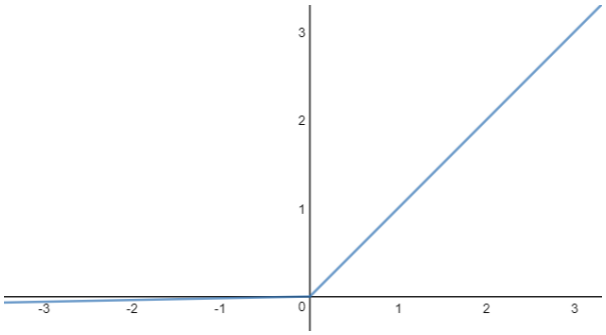


Fig. 15. Función de activación: *Leaky* ReLU.

- *Softmax.* Especialmente útil para tareas de clasificación, esta función suele emplearse en la última capa del modelo. Su objetivo es determinar la distribución de probabilidad de la entrada de pertenecer a las distintas categorías. Se define como:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad para\ j = 1, .., K$$

1.5.5. *Overfitting* o sobreajuste

Uno de los problemas más frecuentes que se encuentran al entrenar una red neuronal, es el sobreajuste u *overfitting.* Consiste en ajustar excesivamente bien los hiperparámetros, parámetros que se fijan antes de realizar el entrenamiento, de la red al set de datos que empleamos como entrenamiento. Este sobreajuste conlleva una mala adquisición de características generales a todos los datos, y, por tanto, reduce la calidad de respuesta ante el conjunto de datos de prueba y ante datos futuros.
Para entender la importancia de reducir este error, se propone el siguiente ejemplo:

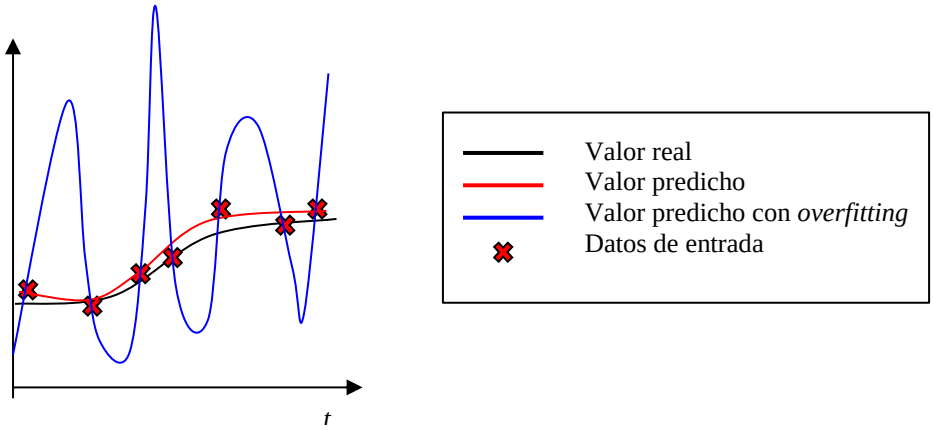


Fig. 16. Ejemplo de sobreajuste.

En la figura 16 se puede apreciar que, con la función azul, la precisión del modelo en el entrenamiento es cercana al 100%. Sin embargo, a la hora de predecir nuevos datos que estén fuera del conjunto de entrenamiento, la precisión del modelo se reduce drásticamente, puesto que solo es preciso en las cercanías al conjunto de entrenamiento.
En cambio, con la función roja, la precisión obtenida durante el entrenamiento será menor, pero la red predecirá con mayor acierto nuevos datos.

El sobreentrenamiento se puede evitar empleando diversos métodos: conjunto de validación, regularización y *dropout*.

1.5.6. *Dropout* o descarte

El *dropout* es un método de regularización que penaliza la función *loss* evitando que las neuronas detecten sets de características dependientes de otras neuronas.
El funcionamiento de esta herramienta consiste en obviar una fracción aleatoria *p* de nodos pertenecientes a una capa oculta, en cada muestra de entrenamiento y en cada iteración. De esta forma, se eliminan tanto la neurona elegida aleatoriamente, como sus conexiones con las capas anterior y posterior, y sus activaciones. Durante la fase de validación, se usan todas las activaciones de la red neuronal, pero reduciéndose por un factor *p*, para tener en cuenta el descarte hecho durante el entrenamiento.
Una de las contrapartidas de incluir esta función, es que el modelo necesita más iteraciones para alcanzar una convergencia. Sin embargo, cada iteración se ejecuta en un tiempo más reducido, y las características extraídas son más robustas.

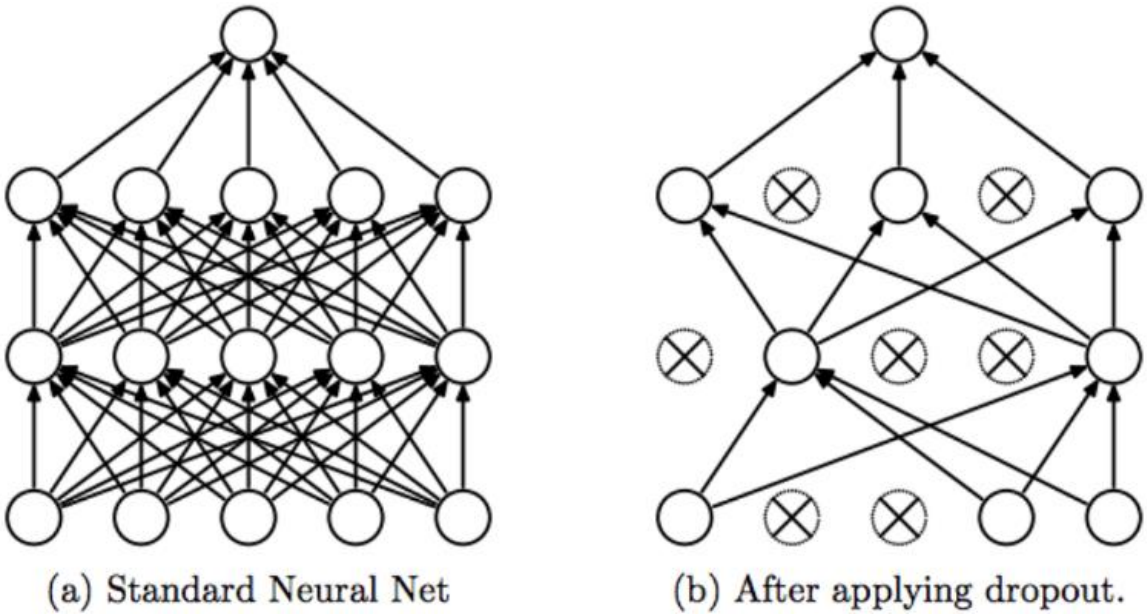


Fig. 17. Ejemplo del funcionamiento del *Dropout* [32].

1.5.7. *One-hot Encoding*

En tareas de clasificación de datos en categorías, cada muestra del conjunto de datos va acompañada por una etiqueta que determina la categoría a la que pertenece. En numerosas ocasiones, esta etiqueta es una palabra (‘perro’, ‘gato’, ‘ave’). Sin embargo, muchos algoritmos de aprendizaje automático solo son capaces de trabajar con datos numéricos.

Una solución es asignar a cada categoría un número entero diferente, empezando por '0' hasta N-1 clases. Esta técnica se conoce como *integer encoding*. La contrapartida es que los algoritmos entienden que un número mayor implica que esa categoría es más importante. Ésto puede conllevar predicciones muy poco certeras y problemas con el aprendizaje.

TABLA I
EJEMPLO DE *INTEGER ENCODING*

Etiqueta	**Valor categórico**
Perro	0
Gato	1
Ave	2
Gato	1

Aparece entonces la codificación *one-hot*. En lugar de asignar un número entero a cada clase, le atribuye un valor binario en función de su pertenencia a la clase. De esta forma, todas las categorías tienen la misma importancia para el modelo.

TABLA II
EJEMPLO DE *ONE-HOT ENCODING*

Perro	**Gato**	**Ave**
1	0	0
0	1	0
0	0	1

1.5.8. Red convolucional

En los últimos años, las redes convolucionales han destacado por su gran eficacia a la hora de resolver problemas de clasificación de imágenes **¡Error! No se encuentra el origen de la referencia.**. Se componen, generalmente, de tres tipos diferentes de capas, cada una acompañada de una función de activación. Estas capas son convolucionales y de *pooling*, y en la parte superior del modelo, se añaden capas de neuronas *fully-connected*. Para disminuir el riesgo de sobreentrenamiento de las neuronas, es común añadir entre estas capas un *dropout*.

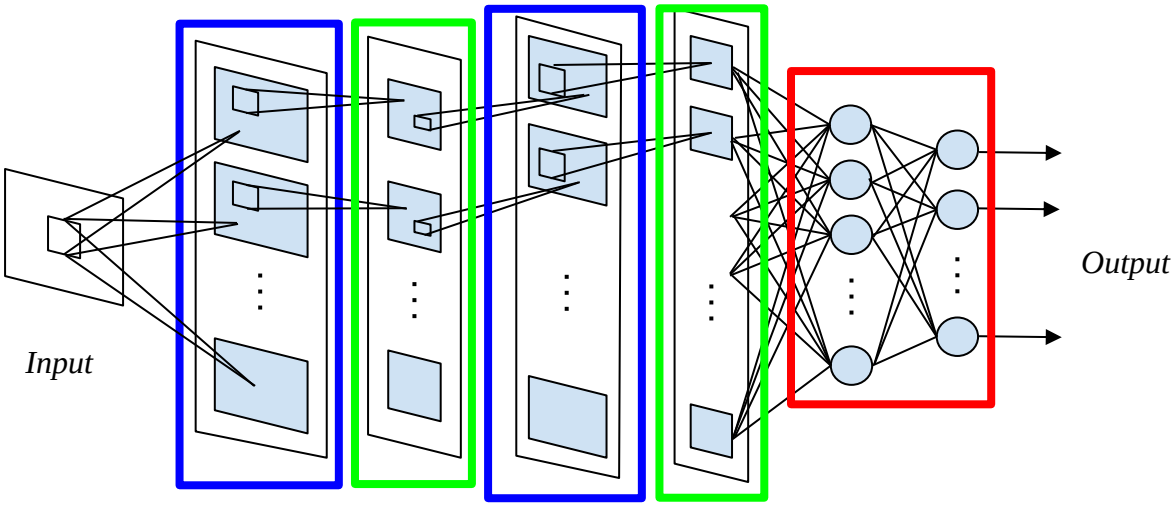


Fig. 18. Red convolucional: capas convolucionales (azul), *pooling* (verde), y *fully-connected* (rojo).

A continuación, se explican con más detalle las distintas capas que forman los modelos convolucionales.

1.5.8.1. Capa convolucional

El núcleo de una red neuronal convolucional es, tal y como su nombre indica, la capa convolucional. Este tipo de capa funciona como un extractor de características. Para llevar a cabo esta extracción, recorre una imagen con distintos filtros, captando cada uno una característica diferente. Cada filtro lleva a cabo una transformación lineal distinta sobre la imagen de entrada.

Una imagen puede entenderse como una matriz de números, en la que cada píxel es un elemento de la matriz. De esta forma, una imagen en color, que tiene tres canales (rojo-verde-azul), puede entenderse como un conjunto de tres matrices de dos dimensiones apiladas. Mediante el operador convolucional, se aplica a submatrices de la entrada, otra matriz, denominada filtro o kernel, que lleva a cabo una transformación lineal de la submatriz a la que se aplica. Dicho filtro se va deslizando por toda la matriz de entrada, desplazándose un número que establecemos de píxeles, o *stride*. La matriz que se genera al aplicar un filtro a toda una imagen o matriz de entrada, se denomina mapa de características. Las dimensiones de este mapa pueden calcularse en función de las dimensiones de la entrada (W y H) y del tamaño del filtro (F_W y F_H), aplicando las fórmulas siguientes:

$$output\ width\ = \frac{W-F_W+2P}{S_W} + 1 \qquad output\ height\ = \frac{H-F_H+2P}{S_H} + 1$$

donde P representa la cantidad de *zero-padding* que se añade al borde de la imagen, y S_H y S_W indican el *stride* vertical y horizontal.
En caso de no incluir *padding*, las ecuaciones son ligeramente diferentes:

$$output\ width\ = ceil\left(\frac{W-F_W+1}{S_W}\right) \qquad output\ height\ = ceil\left(\frac{H-F_H+1}{S_H}\right)$$

Se aplica la función *ceil()* para asegurar que el resultado obtenido es un valor entero.

El mapa de características obtenido, aprende un total de $F_H \cdot F_W \cdot d \cdot K$ pesos, con K biases, donde d representa el número de canales de la matriz de entrada, y K, el número de filtros empleados.

Imaginemos una imagen de 5x5 cuyos píxeles tienen valores entre 0 y 1 en lugar de entre 0 y 255 para simplificar los cálculos. Diseñamos un filtro de 3x3 con las mismas características. En la figura 19, se puede ver cómo se va creando el mapa de características (con fondo rosa), cuando se desliza el filtro (amarillo) a lo largo de la imagen (azul) con un *stride* de 1.

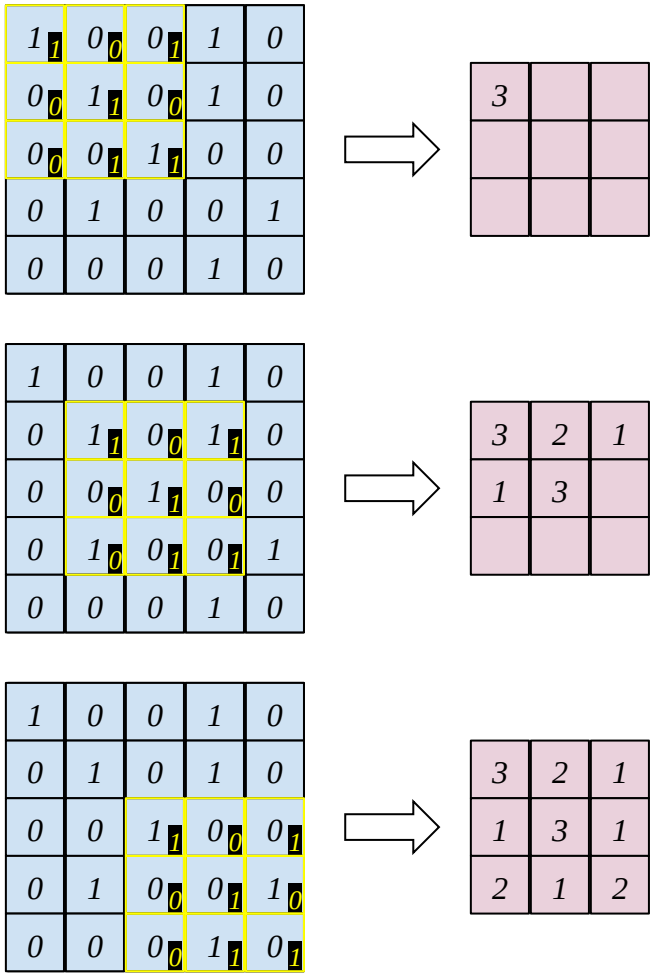


Fig. 19. Funcionamiento de una capa convolucional.

En el caso de imágenes a color, las dimensiones del filtro se ajustan automáticamente al número de canales que tiene la entrada, de forma que transforma cada canal de forma independiente a los demás, y no afecta a la profundidad de la imagen.

Los hiperparámetros a tener en cuenta para establecer las dimensiones del mapa de características son los siguientes [34]:

- Profundidad: hace referencia al número de filtros que se van a aplicar a la imagen de entrada.
- *Stride*: indica la cantidad de píxeles que se desplaza un filtro sobre la matriz de entrada. Cuanto mayor sea el *stride*, menor será el tamaño del mapa de características que se produce.
- *Padding*. Al aplicar un filtro, el píxel que se queda con el valor después de la transformación es el que se encuentra en el centro del filtro (por eso suelen tomarse filtros de tamaño impar). Por este motivo, si se quieren tener en cuenta los datos que se encuentran en los bordes de la imagen, se debe usar una convolución ancha (*wide convolution*), en la que se rellenan los píxeles necesarios con ceros (*zero-padding*) para poder aplicar el filtro. En caso de no usar esta técnica de rellenado, se aplica una convolución estrecha (*narrow convolution*). La convolución ancha es especialmente útil si se tienen filtros de gran tamaño comparado con las dimensiones de la matriz de entrada. Además, eligiendo este modelo de convolución se puede conseguir un mapa de características con las mismas dimensiones que la matriz de entrada.

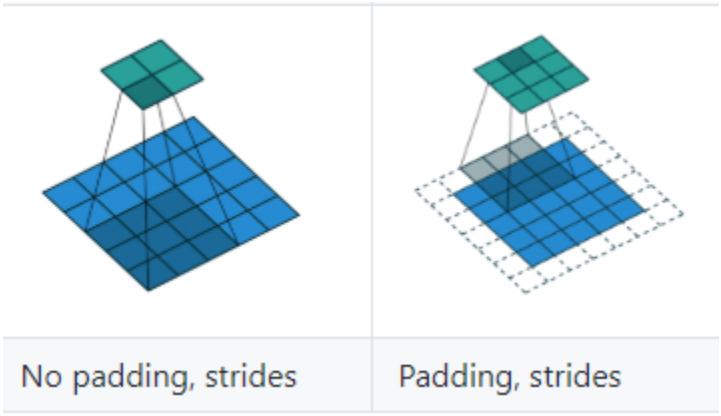


Fig. 20. *Narrow convolution* Vs. *Wide convolution* [35].

Finalmente, la fórmula para calcular la salida de una capa convolucional que recibe como entrada una imagen con *d* canales, donde se utiliza un filtro *K*, un *bias b* y una función de activación σ, es la siguiente:

$$conv(I,K)_{xy} = \sigma(b + \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k=1}^{d} K_{ijk} \cdot I_{x+i-1,y+j-1,k})$$

1.5.8.2. Pooling

Empleada en alternancia con las capas convolucionales, esta capa pretende reducir las dimensiones espaciales sin afectar a la profundidad del modelo o número de canales. Para conseguir su objetivo, desliza un filtro con un tamaño f por la entrada, desplanzándolo un número s de bits a cada vez. Las dimensiones finales de salida se pueden calcular aplicando las siguientes fórmulas:

$$h \equiv Altura\ de\ la\ entrada \qquad hh \equiv Altura\ de\ la\ salida$$

$$w \equiv Ancho\ de\ la\ entrada \qquad ww \equiv Ancho\ de\ la\ salida$$

$$d \equiv Profundidad\ de\ la\ entrada \qquad dd \equiv Profundidad\ de\ la\ salida$$

$$hh = 1 + \frac{h-f}{s} \qquad ww = 1 + \frac{w-f}{s} \qquad dd = d$$

La técnica de *pooling* más utilizada actualmente se denomina *max pooling*. El filtro de esta técnica guarda el valor máximo de los que analiza cuando se aplica.

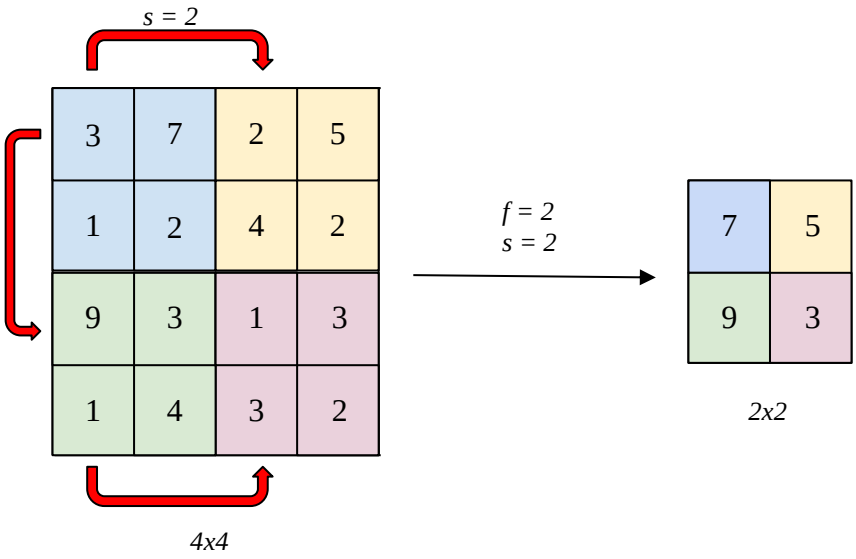


Fig. 21. Ejemplo de *max pooling*.

Otra forma de *pooling* es el *average pooling*, el cual en lugar de mantener el valor máximo, pasa a la salida una media aritmética de los valores que contiene la ventana de la entrada que analiza cada filtro.

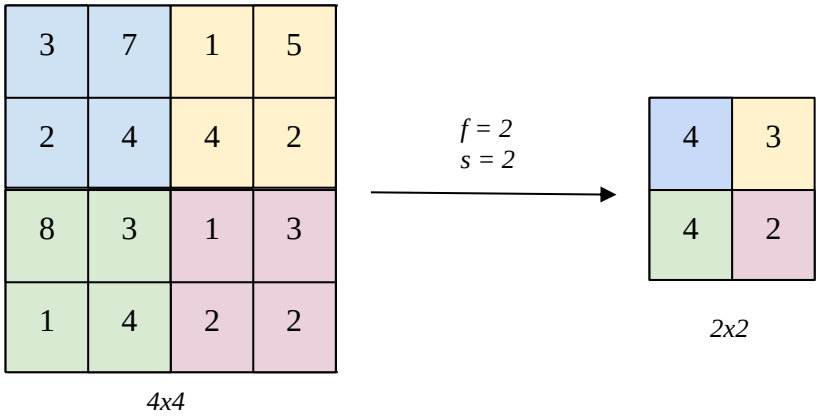


Fig. 22. Ejemplo de *average pooling*.

Independientemente del tipo de *pooling* que se emplee, es importante destacar que esta capa no introduce nuevos parámetros de aprendizaje. Además, al reducir las dimensiones espaciales, disminuye el tiempo de computación necesario y ayuda a prevenir el *overfitting.*

1.5.8.3. Capa *fully-connected*

Para completar una red neuronal convolucional y, en tareas de clasificación de imágenes, para obtener una matriz de probabilidades de pertenencia a cada una de las categorías establecidas, se añaden una o varias capas de neuronas completamente conectadas al final del modelo. Al contrario que las neuronas de una capa convolucional, los nodos pertenecientes a este tipo de capa están conectados a todas las funciones de activación de la capa anterior, sin interactuar con otras neuronas del mismo nivel.

2. Métodos

2.1. Base de datos

La base de datos empleada en este proyecto es la adquirida por Kaneshiro et al. en [36].

2.1.1. Participantes y estímulos

El grupo de sujetos participantes en la toma de datos es muy heterogéneo. De los 10 participantes, con edades comprendidas entre 21 y 57 años, uno es zurdo y siete son hombres. Por tanto, no se limita el estudio de los datos a una población localizada.

Cada individuo ha participado en dos sesiones separadas entre sí por 6-8 días. Cada sesión consiste en tres bloques de 864 muestras cada uno, realizando descansos entre ellos para comprobar la impedancia de los electrodos. Cada muestra equivale a la presentación de una imagen durante 500 ms, con un descanso de 750 ms antes de presentar la siguiente imagen. Cada 36 muestras, se realiza una pequeña pausa.

Las imágenes empleadas como estímulos pertenecen a la base de datos empleada en estudios de Análisis de Similitud Representacional (RSA) [19][20]. De este set de imágenes, tan solo se toman 72 ejemplos, para tener el mismo número de estímulos en todas las categorías. Como se aprecia en la figura 23, las imágenes pueden dividirse en dos grandes grupos: seres vivos e inertes (*animated/inanimated*). Así mismo, los seres vivos pueden ser humanos o animales, y caras o cuerpos. Los objetos inertes son frutas/vegetales u objetos fabricados por el hombre. Cada una de estas categorías cuenta con 12 fotografías que son presentadas en orden aleatorio al sujeto un total de 72 veces. Esto conlleva la presentación de aproximadamente 5184 estímulos a cada participante.

Con el fin de reducir la aparición de artefactos oculares, tanto las imágenes como las pantallas grises de descanso entre estímulos, cuentan con una cruz blanca en su centro, disminuyendo el movimiento de los ojos. Los bordes de colores que aparecen en la figura 23 no se incluyen durante la presentación de los estímulos.

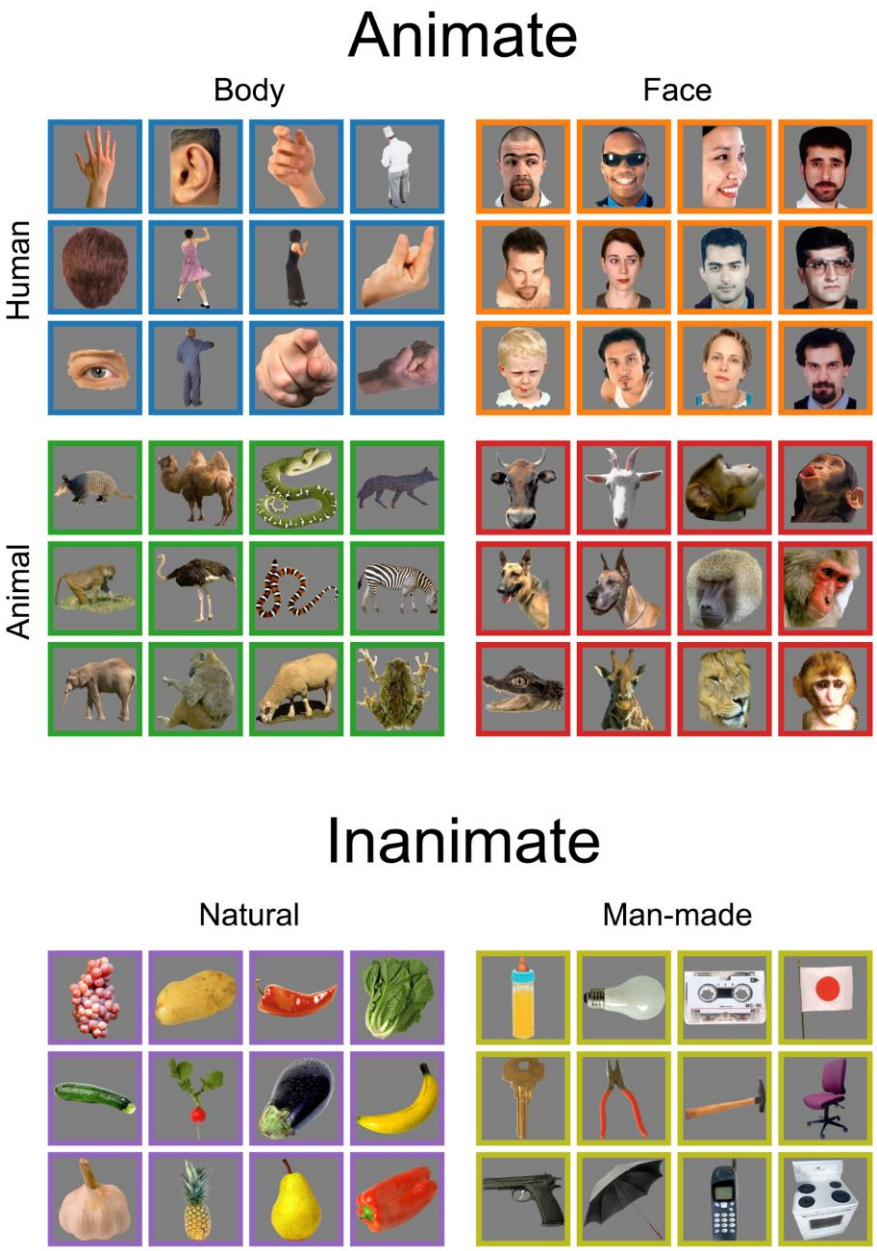


Fig. 23. Set de imágenes empleadas como estímulos durante el experimento de [21].

.

2.2. Adquisición y preprocesamiento de los datos. Equipo empleado

Para la adquisición de las señales electroencefalográficas, se ha utilizado la red de sensores geodésicos HCGSN 110 de la marca EGI. Este dispositivo cubre la totalidad de la cabeza del sujeto, siguiendo el esquema alternativo al internacional 10-20 (figura 24), teniendo un especial cuidado en mantener una distancia adecuada entre los electrodos, y envolviendo también la parte inferior de la cabeza. Estos dos aspectos son fundamentales para obtener una buena calidad y precisión en las lecturas de las señales eléctricas. Los electrodos empleados usan almohadillas y una solución salina en lugar de geles y pegamento, para mejorar la transmisión de la señal al equipo de registro y evitar abrasiones en el cuero cabelludo. Para recoger las lecturas del encefalograma, junto a la red de sensores, se ha empleado el amplificador EGI Net Amps 300 y el software de adquisición EGI Net Station 4.4.

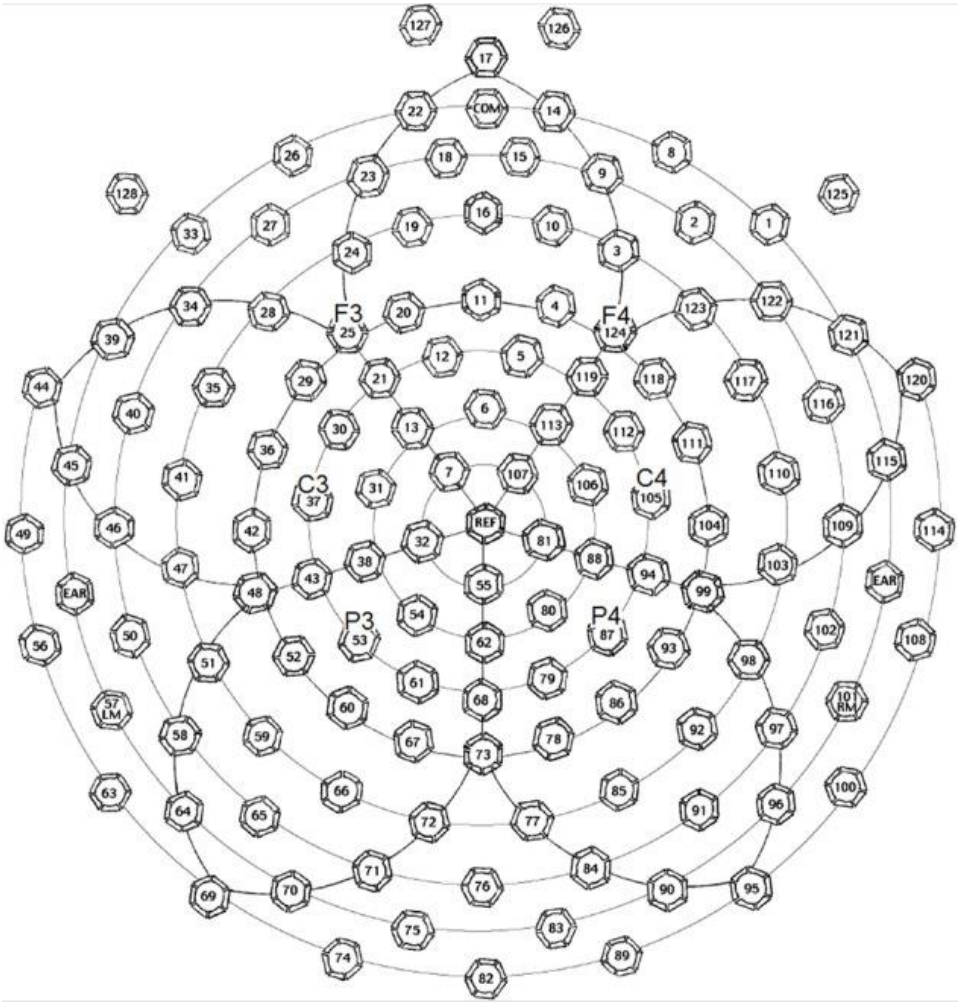


Fig. 24. Posicionamiento de los electrodos en la red de sensores HCGSN 110 de EGI [37].

Fig. 25. Red geodésica de sensores HCGSN [38].

En el experimento conducido por Kaneshiro et Al., se formó una red con un total de 128 canales. Los datos recogidos por dichos canales tomaron como referencia el vértex craneal.

Como se detalla en [21], las señales obtenidas se preprocesaron offline usando el programa de software Matlab. En primer lugar, se aplicó un filtro de paso alto de tipo Butterworth de orden 4, para eliminar todas las frecuencias por debajo de 1 Hz, suprimiendo la componente continua de las señales. Posteriormente se aplicó un filtro

de paso bajo de orden 4 de modelo Chebyshev tipo I para descartar las frecuencias superiores a 25 Hz y se submuestrearon las señales con un factor de 16, para obtener una frecuencia de muestreo final de 62.5 Hz.
A continuación, las señales procedentes de los electrodos 1 al 124 se reservaron para su análisis posterior, mientras que los electrodos VEOG y HEOG se utilizaron para eliminar los artefactos presentes en los datos. Concretamente, para la supresión de los artefactos oculares, se hizo uso del algoritmo Infomax ICA [39].
Finalmente, las señales procesadas y limpias se dividieron en 32 muestras de tiempo (496 ms) post-estímulo y se concatenaron en una única matriz de tres dimensiones en la que las filas se correspondían con los electrodos, las columnas representaban las muestras de tiempo y la profundidad indicaba las pruebas hechas a cada sujeto.
Todos estos datos fueron recopilados en un fichero .mat (Matlab) bajo diversas variables:

- N: Número de muestras de tiempo por prueba. Este valor siempre es igual a 32.
- Fs: Frecuencia de muestreo de los datos. Siempre es igual a 62,5 Hz.
- T: Número de pruebas realizadas. Tiene un valor de aproximadamente 5184 pruebas por sujeto.
- X_2D: Matriz de dos dimensiones representando las muestras en el número de pruebas y las muestras en el tiempo de todos los electrodos concatenados. Sus dimensiones para un sujeto son: (T, 124*N)
- X_3D: Matriz tridimensional anteriormente mencionada con dimensiones: (124, N, T)
- categoryLabels: Vector que incluye la categoría a la que se corresponde cada prueba. Cada número se corresponde con una categoría siguiendo la tabla III. Sus dimensiones son: (T,1).

TABLA III
REPRESENTACIÓN NUMÉRICA DE LAS CATEGORÍAS

Categoría	**Representación numérica**
Cuerpo humano	0
Cara humana	1
Cuerpo animal	2
Cara animal	3
Fruta/Vegetal	4
Objeto inanimado	5

En este proyecto se ha empleado la matriz de datos X_3D y la variable ‘categoryLabels’. En el caso de analizar la precisión de la clasificación de imágenes de los modelos empleando todos los sujetos simultáneamente, se ha procedido a concatenar las matrices de los sujetos en la dimensión del número de pruebas T. Además, para algunos de los experimentos, esta nueva matriz concatenada ha sido mezclada de forma aleatoria en la dimensión anteriormente mencionada. De esta forma, los datos procedentes de todos los sujetos se encuentran entremezclados, buscando un enfoque

más genérico. Así mismo, las etiquetas que acompañan a cada prueba han sido mezcladas en el mismo orden que la matriz de datos y procesadas con la función *one-hot encoding*.

2.3. Entorno de desarrollo

2.3.1. Librerías Tensorflow y Keras

En cuanto al lenguaje de programación empleado para desarrollar las redes neuronales artificiales, se ha utilizado el lenguaje Python, conocido por su gran versatilidad y número de librerías. Entre dichas librerías, se encuentra la librería Keras, especializada en algoritmos de aprendizaje profundo y redes neuronales artificiales.

A su vez, Keras puede funcionar con tres librerías base o *backend*: Theano, la más antigua; Tensorflow, la más eficaz en tareas de clasificación de imágenes; y CNTK, la más rápida en casos de procesamiento de datos, texto y voz. En el proyecto se emplea el *backend* de Tensorflow.

Esta librería permite:

- Crear modelos secuenciales o de tipo *Model()*.
- Añadir diferentes tipos de capas. Por ejemplo: *Conv2D()*, del conjunto *convolutional layers*; *Dropout()*, de la clase *core layers*; o *MaxPooling2D()*, de la categoría *pooling layers*.
- Compilar el modelo creado definiendo el tipo de función de pérdida a emplear, el optimizador utilizado (*Adam* en este proyecto) y los parámetros para evaluar el rendimiento del modelo.
- Entrenar el modelo mediante la función *fit()*. En esta función se deben especificar los conjuntos de entrenamiento y validación, así como el número de iteraciones que se deben hacer durante el entrenamiento.
- Generar predicciones sobre nuevos datos con la función *predict()*.

2.3.2. Google Colaboratory

El ordenador empleado para la consecución de este proyecto cuenta con un procesador Intel Core i3-4030U con 4 GB de RAM. Esta capacidad de procesamiento es insuficiente para poder llevar a cabo el desarrollo de diversos modelos de redes neuronales, por lo que se ha empleado el cuaderno online de Google Colaboratory [40][40].

Google Colaboratory es una plataforma online de la empresa Google, que pone a disposición de los usuarios una GPU gratuita Tesla K80.

Fig. 26. Nvidia Tesla K80 [41].

Se trata de un programa de investigación que busca facilitar el acceso a un entorno de aceleración para los desarrolladores de inteligencia artificial. Cuenta con una memoria RAM limitada a 11.17 GB, pero permite una ejecución de la librería Tensorflow hasta 45 veces más rápida que la realizada mediante una CPU[42].

TABLA IV
TIEMPOS DE EJECUCIÓN CON CPU Y GPU

Acelerador por hardware	**Tiempo**
Ninguno (CPU)	606 us/step o 3s/epoch
GPU Tesla K80	230 us/step o 1s/epoch

La tabla IV muestra una comparativa de los tiempos de ejecución de una red neuronal artificial sencilla, compuesta únicamente por una capa densa oculta de 256 neuronas, y otra de salida con 6 neurona. Al complicar el modelo, la diferencia en el tiempo empleado en función de si se emplea un acelerador por hardware o no, se amplía.

2.3.3. Software desarrollado

El código implementado en este proyecto, puede encontrarse en el siguiente enlace: https://github.com/Angordil/Analisis-de-registros-de-EEG-mediante-redes-neuronales.git

En dicho enlace se encuentran los códigos completos necesarios para reproducir los resultados que se mostrarán en esta memoria en un epígrafe posterior (3). Se compone de un archivo *Readme* desde el que se puede descargar la base de datos empleada y creada en [36]. Además, hay dos archivos de preparación de datos en los que se extraen de los ficheros .mat las matrices en dos dimensiones que contienen la información sobre los electrodos y las lecturas de cada uno, así como las etiquetas que acompañan a cada muestra. Se realiza un pequeño procesamiento consistente en convertir la matriz en una matriz de tres dimensiones, facilitando la visualización de los datos. Además, se aplica la codificación *one-hot* a las etiquetas para que el sistema otorgue la misma importancia a todas ellas.

Por otro lado, se incluyen también los códigos correspondientes a cada modelo propuesto a continuación.

2.4. Modelos propuestos

Se han desarrollado dos tipos de redes: simples y complejas. Las redes sencillas cuentan con un máximo de 7 capas profundas, siendo éstas únicamente de neuronas completamente conectadas, o una combinación de convoluciones, capas de reducción de dimensiones y capas densas. La combinación de varias de estas redes lleva a la formación de los modelos más complejos: el modelo de votación, y el de validación cruzada.

2.4.1. *Dense*

Este modelo es el más sencillo de los que se proponen. Se compone únicamente de tres capas profundas *dense* o *fully-connected*. La última de ellas cuenta con solo seis neuronas, debido a que se desea clasificar las imágenes en una de las seis categorías mencionadas con anterioridad. Las otras dos capas densas funcionan como extractores de parámetros. El número de neuronas se reduce de una capa a la siguiente para ir reduciendo las posibles salidas predichas progresivamente.

Así mismo, para prevenir el sobre entrenamiento de la red, se incluye un descarte aleatorio de un porcentaje de los datos entre cada capa. Se ha probado con tres porcentajes diferentes de descarte (15%, 30% y 50%), obteniéndose los resultados de la figura 27. Se puede apreciar que, con un descarte del 30%, se consigue una precisión alta en el entrenamiento, con un menor sobreajuste de los datos que con un descarte del 50%. La tasa de acierto del conjunto de validación, muy similar en los tres casos, presenta una menor variabilidad en la línea amarilla. Por esto, se elige un *dropout* de 0.3 para el modelo. Además, este valor es el habitualmente usado en aprendizaje profundo.

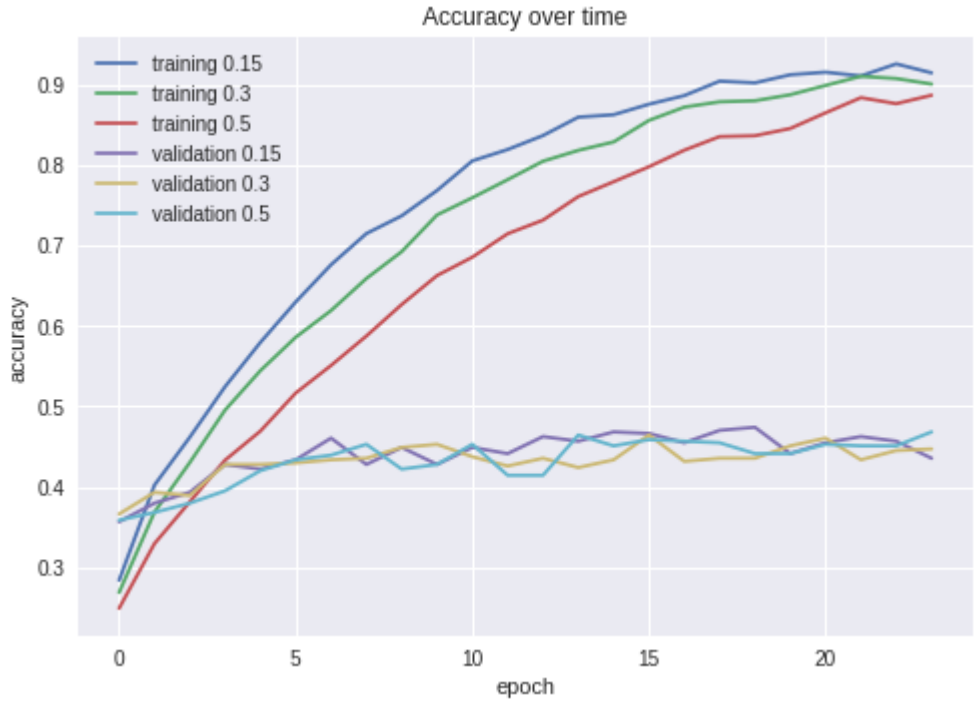


Fig. 28. Comparativa de modelos densos con varios valores de descarte.

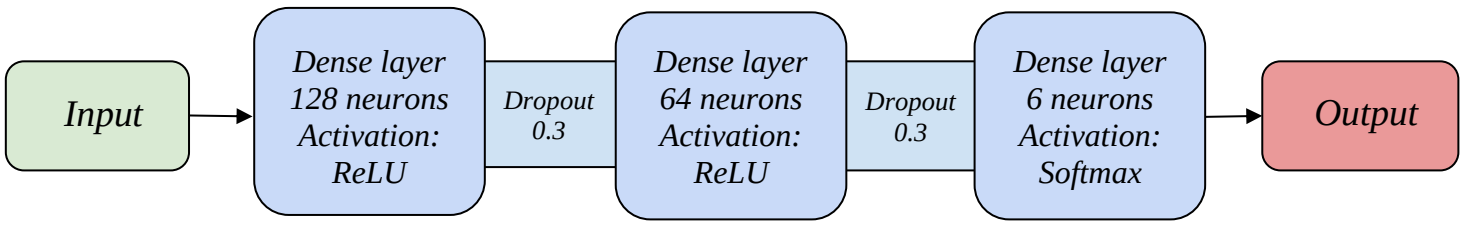


Fig. 27. Esquema del modelo denso.

Tras comparar la precisión de los modelos en clasificar las imágenes durante el entrenamiento al variar los hiper parámetros de las capas *dense* (el número de neuronas) y de la capa de descarte, se ha determinado que la mejor combinación es la que se muestra en la figura 28.

Al emplear un número elevado de neuronas en la primera capa densa, se consigue una mayor precisión del modelo, aunque si este número es demasiado alto, se produce sobreaprendizaje del modelo. Si, por el contrario, la cantidad de nodos es bajo, el modelo requerirá un tiempo mayor para alcanzar la convergencia, siendo esta inferior a la óptima debido a que el modelo puede aprender un número menor de parámetros.

2.4.2. Red convolucional. Una capa Vs. Dos capas convolucionales

En los últimos años, las redes convolucionales han demostrado ser el modelo de red neuronal artificial más apto para resolver tareas de clasificación de imágenes. Es por ello que se ha decidido implementar un modelo sencillo de este tipo de red con la base de datos descrita, aunque el tipo de imagen se corresponda más bien con un mapa de calor en lugar de una fotografía o dibujo.

Para comparar la eficacia de este modelo, se han desarrollado dos redes convolucionales, como se muestran en las figuras 29 y 30, construidas sobre el modelo denso comentado en el apartado anterior. La primera de ellas cuenta solo con una única capa convolucional seguida de una reducción de dimensiones (*Max Pooling*) y un descarte aleatorio de datos. La segunda y ligeramente más profunda, incorpora además una segunda capa convolucional con mayor número de filtros entre la primera reducción de dimensiones y la capa de *dropout*.

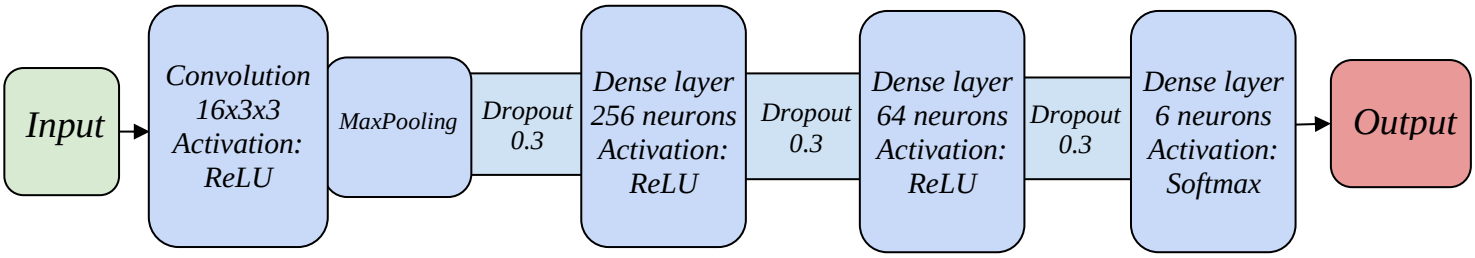


Fig. 29. Esquema del modelo convolucional simple.

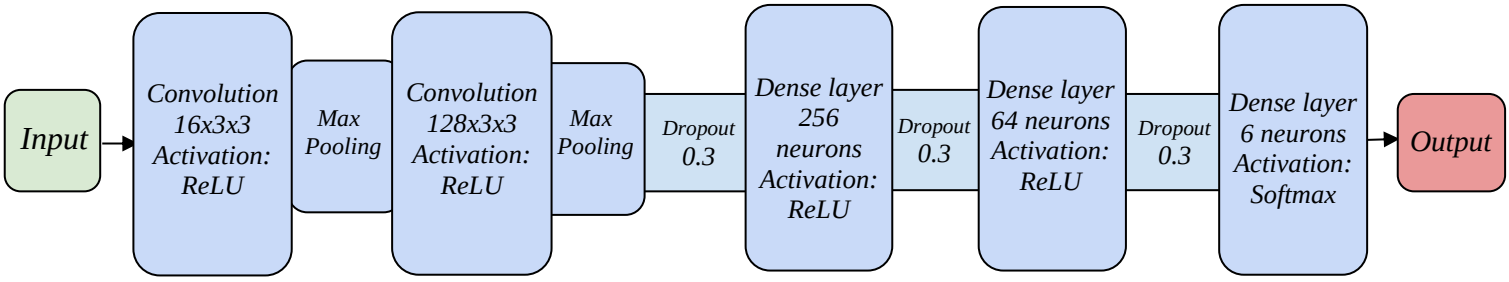


Fig. 30. Esquema del modelo convolucional doble.

2.4.3. Modelo de votación

Este modelo tiene como finalidad comprobar la precisión con la que se puede predecir la imagen que está viendo un sujeto sin haber entrenado anteriormente ninguna red con datos de dicho sujeto. Su funcionamiento es el que se muestra en la figura 31. Se introduce una lectura del nuevo sujeto (en este caso, del sujeto número 10) en nueve modelos que han sido previamente entrenados con los datos de los demás participantes del experimento. Cada modelo genera una predicción de la categoría a la que pertenece la imagen quedándose con la clase que presenta una mayor probabilidad. Finalmente, la categoría escogida como la predicha, es la que han votado los modelos con mayor frecuencia, es decir, la moda de las predicciones.

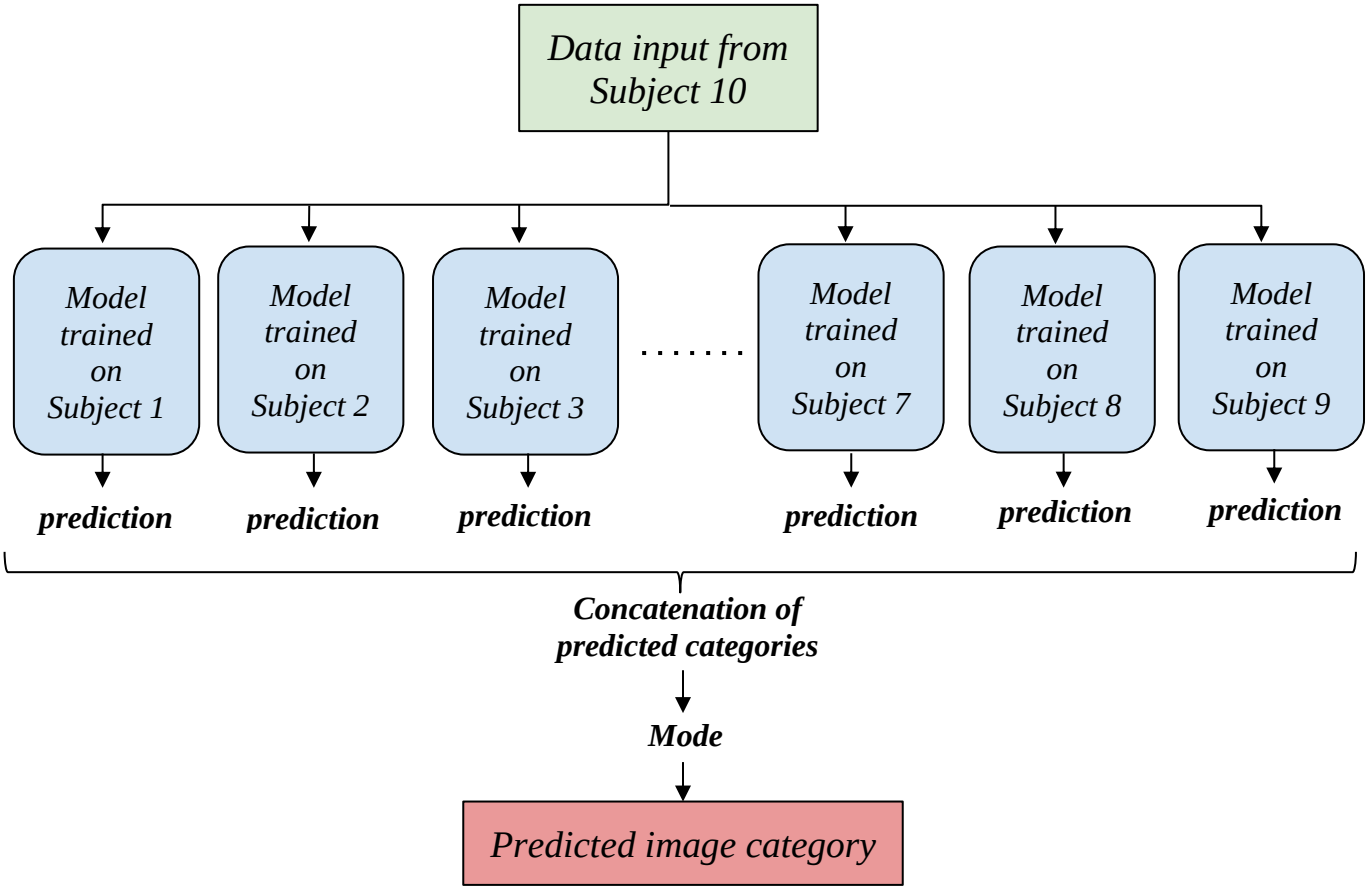


Fig. 31. Esquema del modelo de votación.

2.4.4. Validación cruzada

La validación cruzada o *cross-validation* en inglés, es un método comúnmente utilizado para proporcionar robustez al modelo. Consiste en dividir los datos en subgrupos, empleando una parte de ellos para el entrenamiento de la red y, la restante, para la validación de la misma.

Este método tiene como base la validación simple, en la cual los datos forman dos grupos: uno para la construcción del modelo y, el otro, para la comprobación del error que presenta dicho modelo.

En la validación cruzada, se separa el conjunto de datos en *k*-subgrupos y se llevan a cabo *k* iteraciones del modelo. En cada una de estas iteraciones, *k*-1 subgrupos conforman el conjunto de entrenamiento y el restante representa el conjunto de validación. Finalmente, la media aritmética de las iteraciones, proporciona el ratio de error que presenta la red neuronal.

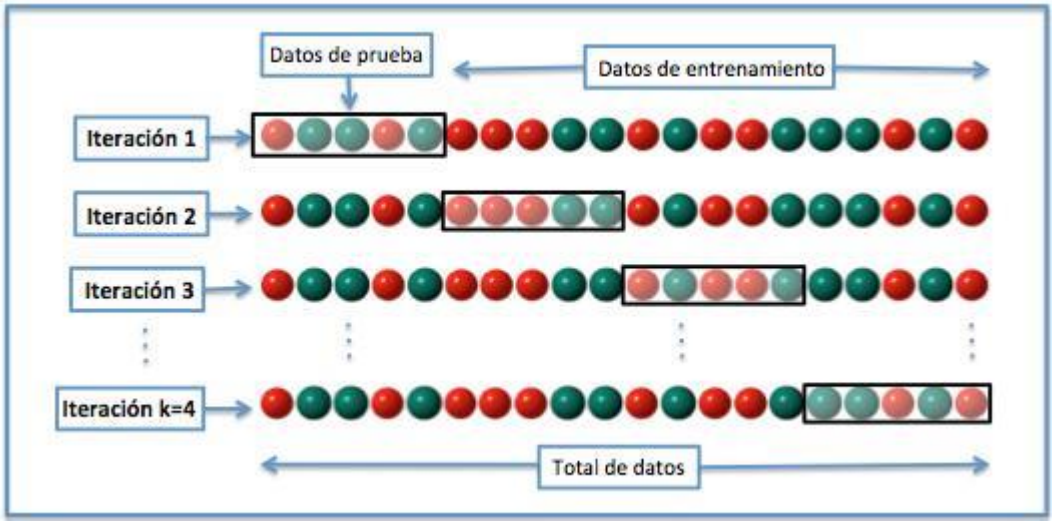


Fig. 32. Validación cruzada [43].

En este proyecto se ha establecido un número de iteraciones $k = 10$. La elección de esta cantidad se debe a que se cuenta con los datos de diez participantes, por lo que, para comprobar la generalización y respuesta de un modelo ante un nuevo sujeto, se emplean como conjunto de entrenamiento los datos de los nueve sujetos restantes concatenados. Por otro lado, este valor también se utiliza en la predicción de nuevos datos de sujetos con los que ya se ha entrenado la red neuronal.

Para finalizar, los modelos entrenados mediante esta metodología y la descrita en el apartado anterior, se han construido tomando como base la ya expuesta red convolucional con dos capas de convolución.

3. RESULTADOS Y DISCUSIÓN

En este apartado se van a exponer los distintos resultados obtenidos durante la experimentación en función de tres supuestos:

1. Introducción de datos nuevos del sujeto con el que se ha entrenado la red.
2. Introducción de datos nuevos de uno de los sujetos con los que se ha entrenado la red.
3. Introducción de datos de un nuevo sujeto.

En el supuesto 1, se toma como sujeto experimental al participante número uno.

Para hacer el análisis de los resultados, se estudiarán la precisión tanto del entrenamiento como de la validación, así como los valores de la pérdida o error de ambos.

- ➢ Supuesto 1

En la figura 33 se pueden observar los resultados obtenidos al ejecutar los tres modelos base en el supuesto 1. En la columna de la izquierda se representa la precisión de la red al clasificar las imágenes tanto en el entrenamiento (en azul) como durante la validación (en verde). En los tres casos se puede ver como se produce un sobreajuste de las redes a los datos de entrenamiento, obteniéndose tasas de acierto muy diferentes entre el conjunto de prueba y el resto de los datos.

A medida que se complica la red con mayor número de capas escondidas y de tipo convolucional, el porcentaje de precisión en el acierto disminuye ligeramente, siendo de un 45% para el modelo denso, un 43% para el modelo convolucional con una capa de convolución, y, para el modelo con dos de estas capas, un 41%.

En cuanto a las pérdidas y el error reflejados en las gráficas de la columna central, los valores obtenidos con las tres configuraciones son muy similares.

Observando las matrices de confusión en la columna de la derecha, se aprecia en todos los casos una diagonal más marcada que empieza en la esquina superior izquierda para terminar en la esquina opuesta. Esto implica que, en un gran número de casos, el sistema es capaz de establecer correctamente la categoría con la que se corresponde cada imagen introducida.

En líneas generales, los tres modelos tienen un funcionamiento similar. Sin embargo, atendiendo a la velocidad a la que se produce el sobreentrenamiento de las redes, y los valores de error y precisión, se estima que el modelo convolucional simple, es el que presenta el mejor resultado en caso de introducir datos nuevos del sujeto con el que se entrena la red neuronal artificial.

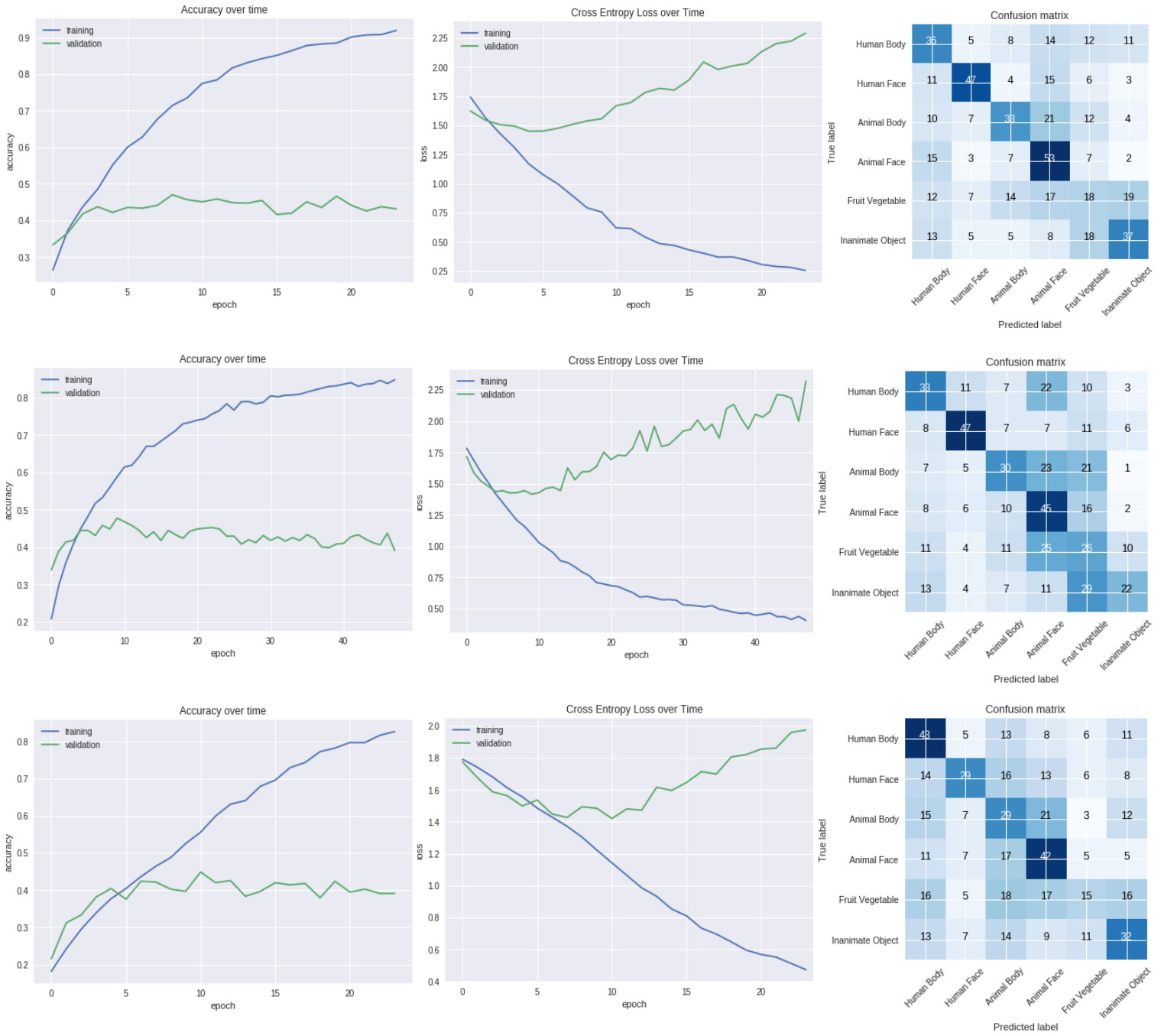


Fig. 33. Precisión frente a tiempo (izquierda), pérdida frente a tiempo (centro) y matriz de confusión (derecha) de los modelos denso (arriba), convolucional simple (medio) y convolucional con dos capas de convolución (abajo) en el supuesto 1.

➢ Supuesto 2

En este segundo caso en el que los modelos se han entrenado con datos de los 10 sujetos participantes, se observa una clara mejora en las tasas de acierto de las tres configuraciones. Esta tasa experimenta un incremento de aproximadamente un 30% respecto a la obtenida en el supuesto anterior.

Sigue apareciendo un cierto sobreajuste de los modelos a los datos de entrenamiento, aunque, en este caso, la diferencia entre los resultados de los conjuntos de entrenamiento y validación no resulta tan grande.

Bajo este supuesto, los tres modelos presentan una actuación muy buena, destacándose los resultados de la red convolucional doble, con una precisión del 72% y el menor error de las tres configuraciones.

En cuanto a las matrices de confusión, la diagonal que representa la correcta clasificación de las imágenes en las distintas categorías se muestra más marcada, indicando un mayor grado de precisión.

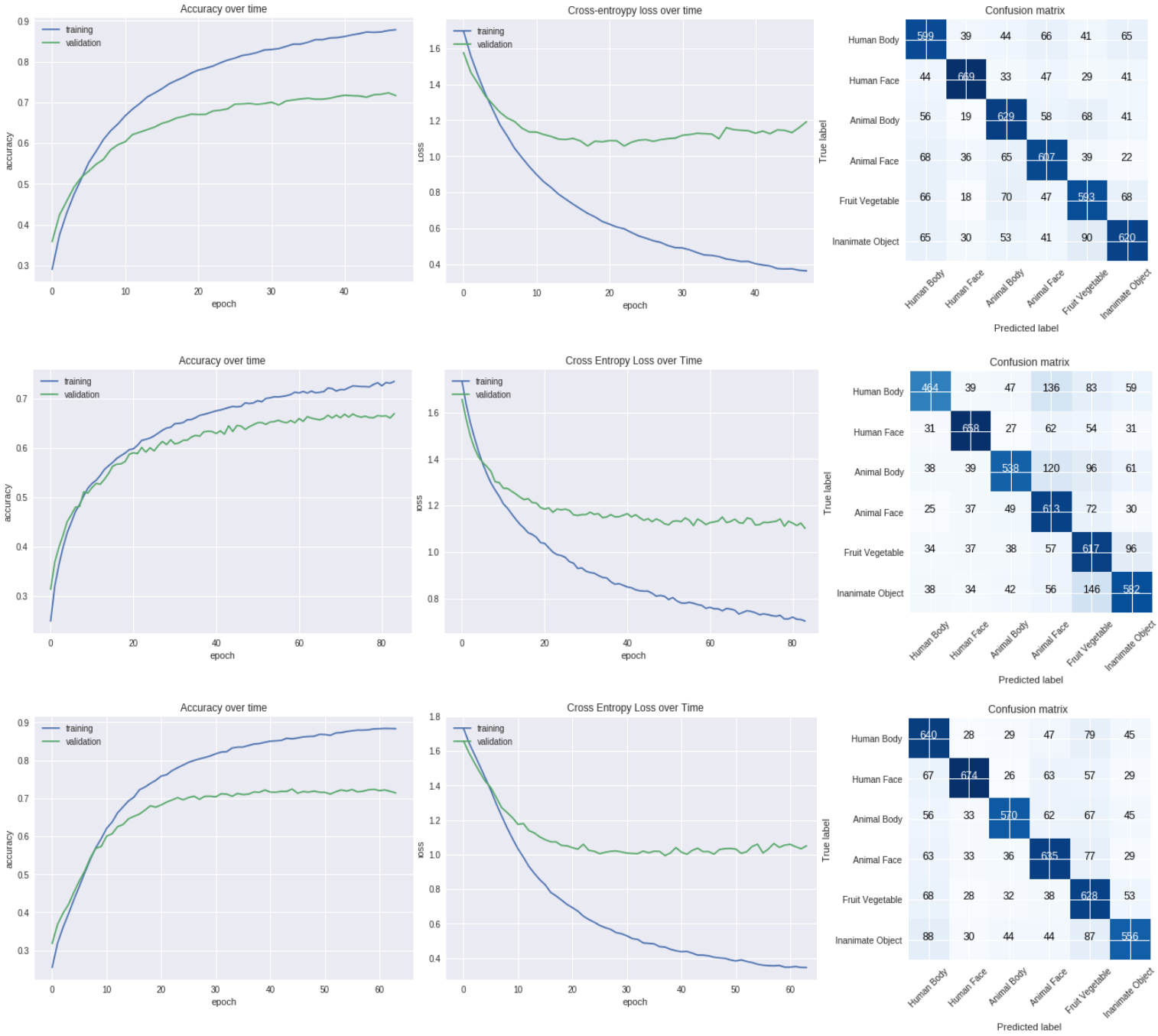


Fig. 34. Precisión frente a tiempo (izquierda), pérdida frente a tiempo (centro) y matriz de confusión (derecha) de los modelos denso (arriba), convolucional simple (medio) y convolucional con dos capas de convolución (abajo) en el supuesto 2.

➢ Supuesto 3

Finalmente, en caso de entrenar los modelos con los datos de nueve de los sujetos participantes en el experimento, y realizar la validación con las lecturas del décimo sujeto, se obtienen los resultados de la figura 35.

El sobreentrenamiento de los datos se aprecia claramente en las tres configuraciones, destacándose en los dos modelos más sencillos.

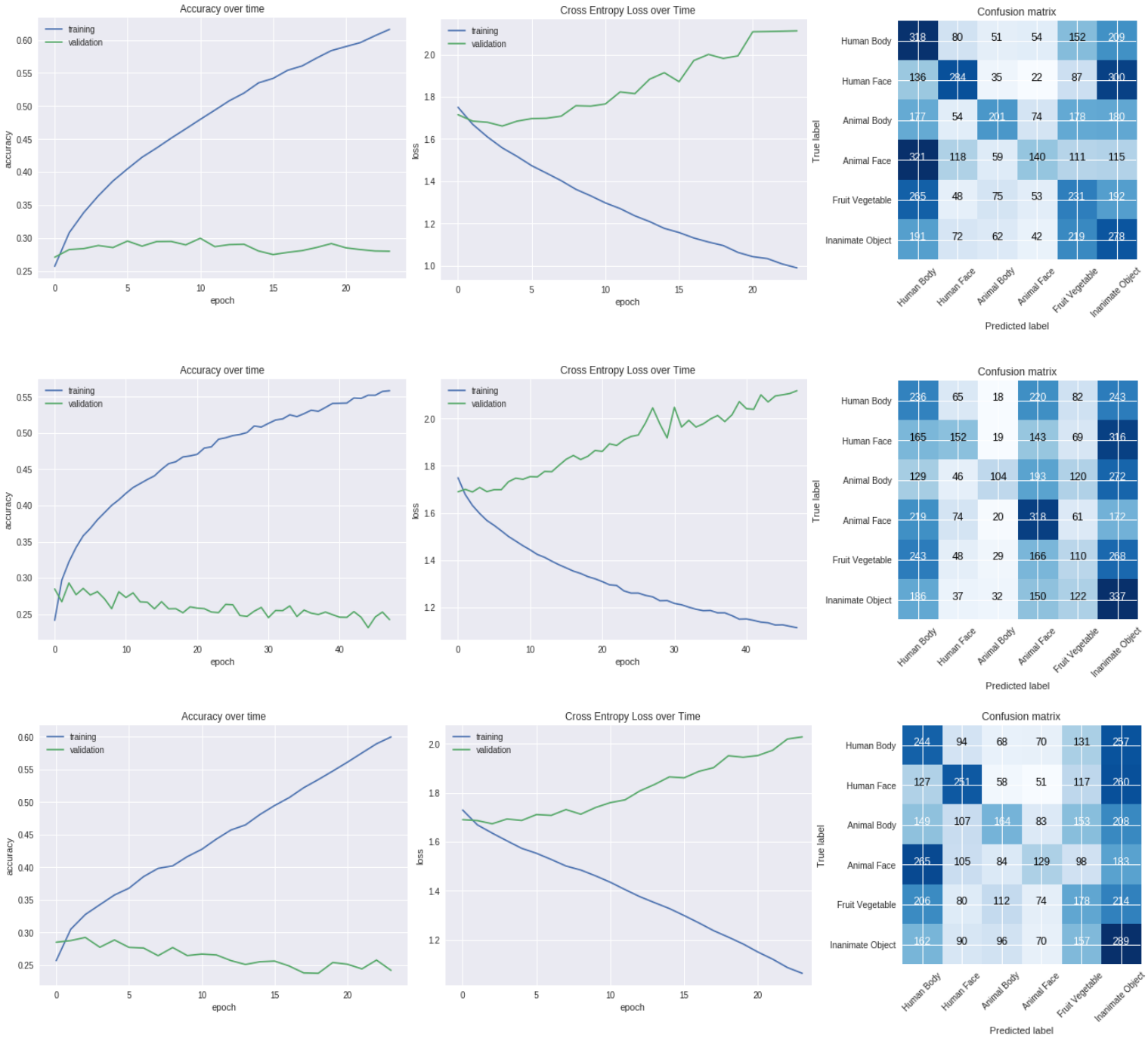


Fig. 35. Precisión frente a tiempo (izquierda), pérdida frente a tiempo (centro) y matriz de confusión (derecha) de los modelos denso (arriba), convolucional simple (medio) y convolucional con dos capas de convolución (abajo) en el supuesto 3.

Al mismo tiempo, al aumentar el número de epochs durante los que se entrenan las redes, la tasa de precisión que presentan va en disminución, obteniéndose aproximadamente un 29% de acierto en el modelo denso, un 27% en el convolucional simple, y un 26% en el convolucional doble.

Finalmente, en este supuesto, la diagonal característica de las matrices de confusión se muestra difusa o inexistente en algunas partes, implicando una tendencia a clasificar incorrectamente los datos introducidos. El modelo denso, por ejemplo, presenta una inclinación por clasificar las imágenes como partes del cuerpo humano, mientras que la red convolucional simple tiende a etiquetarlas como objetos inanimados.

Con el objetivo de mejorar los resultados obtenidos en el supuesto 3, se ha planteado un modelo de votación. De esta forma, en lugar de entrenar una única red con los datos de nueve sujetos para luego validarla con el décimo participante, se entrena un modelo por persona, y después cada red 'vota' la categoría a la que se corresponde la imagen introducida para la validación. La etiqueta que se asigna a dicha imagen es la que recibe el mayor número de votos.

En la figura 36 se muestran los valores de precisión obtenidos en las nueve redes neuronales de las que se compone el modelo de votación. Estas redes siguen la configuración de la red convolucional doble. Como se puede apreciar en el gráfico, las tasas de acierto varían de forma considerable en función del sujeto con el que se entrena. De esta forma, la red del sujeto ocho presenta la menor precisión, mientras que la red construida con los datos del participante número seis acierta en un mayor número de ocasiones.

Finalmente, la probabilidad de clasificar correctamente las imágenes de entrada empleando este modelo de votación, es del 28%.

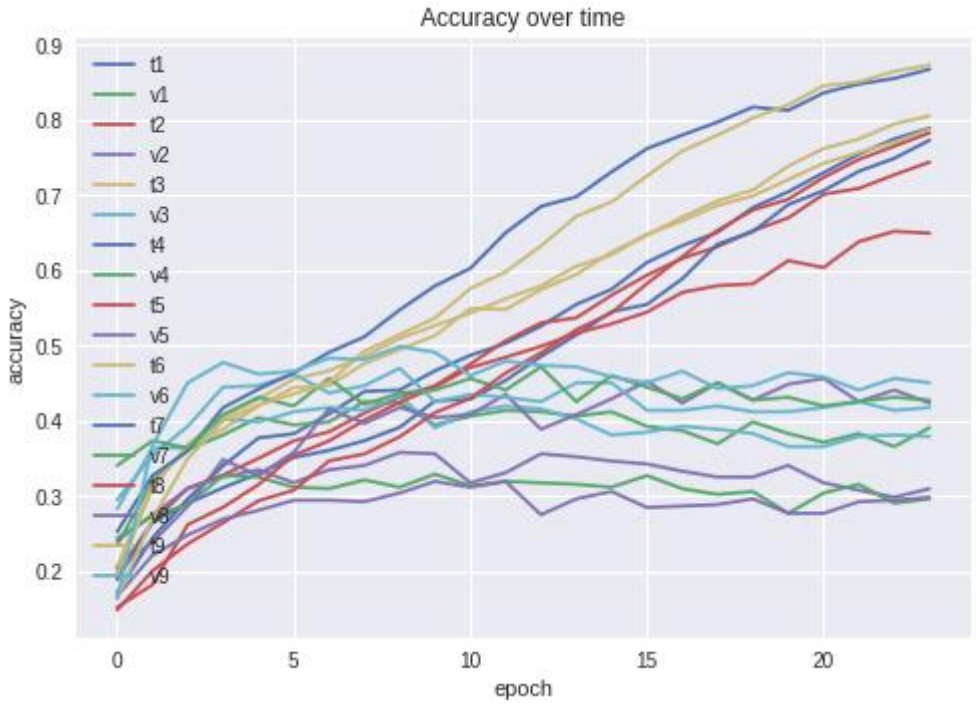


Fig. 36. Precisión en el entrenamiento (rojo, amarillo y azul oscuro), y en la validación (morado, verde y azul claro) de los nueve modelos que se emplean en el modelo de votación.

La gran diferencia que existe entre los resultados obtenidos en los supuesto dos y tres, se debe a que el cerebro de cada persona es diferente. Por este motivo, al entrenar una red con los datos de un sujeto y realizar la validación con los de un sujeto diferente, resulta en una tasa de precisión baja (alrededor del 25%). Sin embargo, si el modelo se entrena con datos de varios sujetos y el conjunto de test se forma con nuevas lecturas de dichas personas, los resultados serán mejores, dado que los parámetros que la red aprende durante el entrenamiento son más generalizados y, por tanto, presentan una buena actuación frente a nuevos casos.

Validación cruzada

Este último método se ha implementado como un intento de optimizar las propuestas comentadas con anterioridad. Por ello, tan solo se ha aplicado a aquellos modelos que presentaban unos resultados mejores. Esta condición se cumple cuando las tres configuraciones básicas se implementan siguiendo las directrices del segundo supuesto.

En la tabla V, quedan recogidas las tasas de acierto obtenidas con las redes básicas en el supuesto dos, aplicando una validación cruzada con 10 iteraciones, de forma que, en cada iteración, se valide la red con un total de 5186 datos y se entrene con los restantes 46671.

TABLA V
TASA DE PRECISIÓN CON EL MÉTODO DE VALIDACIÓN CRUZADA

Modelo	**Precisión**
Denso	71.5 %
Convolucional simple	62.9 %
Convolucional doble	71.5 %

La precisión de los modelos denso y convolucional simple son iguales, pero la convergencia de los datos se alcanza en un menor número de epochs en el caso de la red convolucional, por lo que ésta se considera como la opción óptima.

4. PLANIFICACIÓN

En este epígrafe se van a detallar las fases de las que se ha compuesto el este proyecto de fin de grado, así como el tiempo dedicado a cada una de ellas.

El desarrollo de este proyecto de fin de grado ha pasado por distintas etapas consistentes en:

- Aprender el lenguaje de programación Python y sus librerías especializadas en redes neuronales.
- Entender los conceptos básicos del aprendizaje profundo y de las redes neuronales artificiales (RNAs).
- Estudiar el problema planteado y buscar posibles soluciones.
- Prueba y análisis de las distintas redes propuestas.
- Escritura de la memoria.

En la figura 37 queda representado mediante un diagrama de Gantt el desarrollo en el tiempo del proyecto.

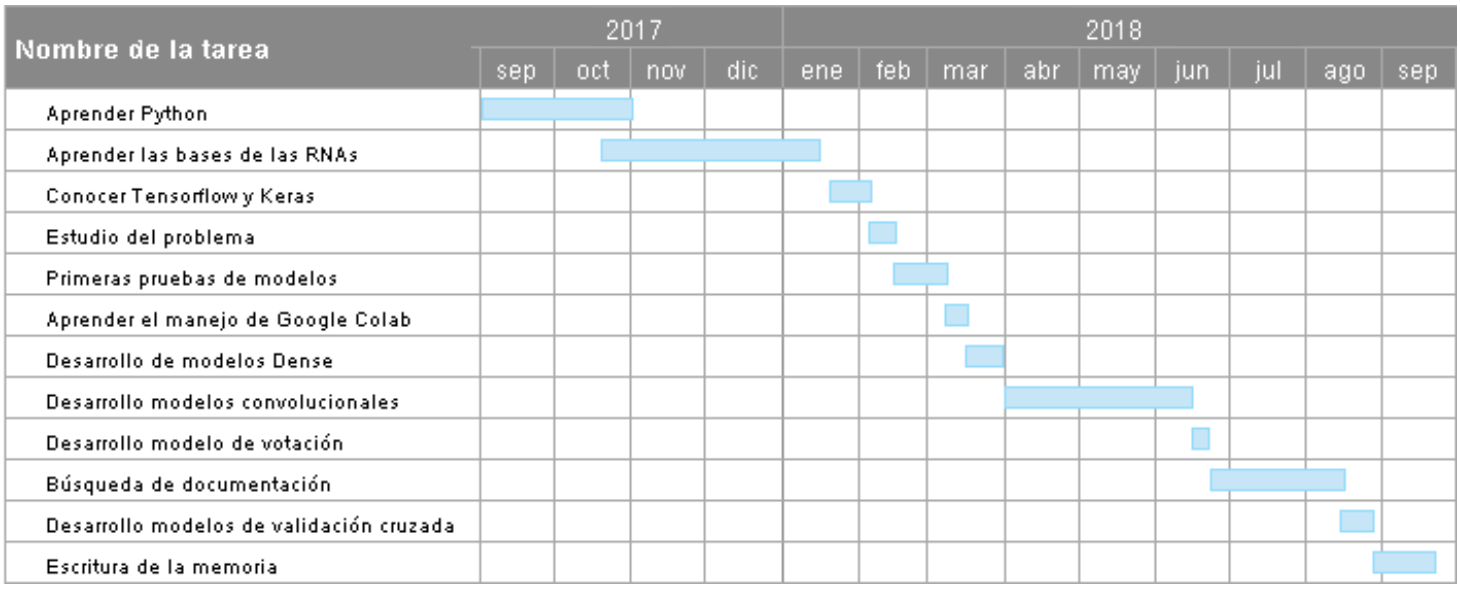


Fig. 37. Diagrama de Gantt.

5. ESTUDIO ECONÓMICO

A continuación, se presentará el presupuesto estimado que se ha requerido para el desarrollo del proyecto. Finalmente, se hará un estudio económico de la posible aplicación y desarrollo desde cero del proyecto al completo, incluyendo la construcción de la base de datos.

5.1. Presupuesto

En cuanto al presupuesto necesario para llevar a cabo este proyecto, se puede dividir en dos partes.

La primera hace referencia a la realización de este proyecto en sí, es decir, del desarrollo de los modelos de redes neuronales. En esta parte han participado una estudiante de grado de ingeniería, y un ingeniero industrial como tutor. El coste derivado de la participación de ambos se resume en:

- Trabajo de la alumna durante 3 horas diarias durante 7 meses, con un coste de, supongamos, 400€ mensuales. Total: 2.800 €
- Reuniones y participación del tutor: 20 horas, con un coste de 60 €/hora. Total: 1.200 €.

Por otro lado, hay que tener en cuenta el software y el hardware empleados durante el proyecto. Se ha utilizado un ordenador portátil Asus TP300L valorado en 250€ y el software libre Google Colaboratory, de coste cero.

El presupuesto total del proyecto queda recogido en la siguiente tabla:

TABLA VI
RESUMEN DEL PRESUPUESTO

Coste de personal	4.000 €
Coste de hardware	250 €
Coste de software	0 €
TOTAL	**4.250 €**

5.2. Posible comercialización del proyecto

En el supuesto de que una empresa quisiera desarrollar este proyecto desde el principio, incluyendo la adquisición de la base de datos, debería hacer frente a los siguientes costes:

- Un equipo de encefalografía completo, incluyendo un casco de electrodos con 128 canales, un amplificador y un software de adquisición de las señales. En el mercado existen diversas opciones:
    - El equipo empleado en [21], que incluye un casco de electrodos HCGSN 110 de 128 canales, con un amplificador Net Amps 300 y un software de adquisición Net Station 4.4, todo ello de la marca EGI. Coste estimado: 15.000 €
    - El equipo completo utilizado en [50], compuesto por el casco de electrodos modular Easy-Cap, un amplificador V-amp de 8 canales de BrainProducts y el software BrainVision Recorder V-Amp edition de la misma empresa. Coste estimado: 12.000 €

- El software de análisis de señales Matlab con el complemento EEGLab, cuya licencia tiene un precio estimado de 120€.
- Los honorarios de un técnico que ejecute los algoritmos desarrollados en este proyecto de fin de grado. Al tratarse únicamente de la aplicación y no del desarrollo de estos, el trabajo del técnico se reduciría a 80h (dos semanas). Si sus honorarios son de 900€/mes, el coste del técnico sería de 450€.
- Los honorarios de los técnicos encargados de la obtención de las señales electroencefalográficas. Coste estimado para un técnico durante las dos sesiones de adquisición de datos (20h): 275€
- La compensación económica a los sujetos voluntarios para la toma de datos, suponiendo 25€/h, dos horas por sujeto y 10 participantes, el coste total es 500€.
- Un ordenador de mesa o servidor con un acelerador por hardware GPU con un mínimo de 20 GB, y un RAM de 8GB. En el mercado existen numerosas opciones que cumplen estos requisitos, con un precio medio de 3.000€.
  Este elemento es optativo, ya que los algoritmos desarrollados pueden implementarse utilizando el software gratuito de Google, Colaboratory.

Sin tener en cuenta la amortización de los materiales empleados en el desarrollo del proyecto, el presupuesto necesario para llevarlo a cabo sería el recogido en la tabla VII.

TABLA VII
PRESUPUESTO DEL ESTUDIO ECONÓMICO

Concepto	**Coste estimado**
Equipo de encefalografía	12.000 – 15.000 €
Técnico analista/programador	450 €
Técnico de laboratorio	275 €
Compensación económica a los voluntarios	500 €
Equipo de procesamiento: ordenador o servidor	3.000 €
Licencias de software	120 €
TOTAL	**16.345 – 19.345 €**

5.3. Marco regulador

La realización del proyecto presentado en esta memoria, no está sujeto a ninguna normativa ya que la base de datos empleada es de origen oficial. En [21], para la obtención de los datos, todos los participantes firmaron un consentimiento previo a las sesiones experimentales. Además, sus datos quedan protegidos, asegurando su completo anonimato, por la siguiente normativa:

- Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos).

Por otro lado, el proyecto desarrollado es de carácter público, por lo que se puede incluir en licencias de código abierto como, por ejemplo, GPL ofrecida por la Free Software Foundation (FSF).

6. CONCLUSIÓN

Empezando por la obtención de los datos y siguiendo todo el camino a través de la preparación de los mismos y la creación y entrenamiento de los modelos propuestos, se puede apreciar que, correlacionar las imágenes que ve una persona con las reacciones que se generan en su cerebro, no es una tarea evidente. Todo el proceso es costoso, y la obtención de los medios para llevarlo a cabo no es fácil.

En este proyecto se ha empleado una base de datos reducida que se obtuvo para otra línea de investigación, con la que se conseguía un porcentaje de acierto en la clasificación de un 40.7% empleando el método de Ánalisis de Componentes Principales (PCA). Se ha conseguido mejorar la tasa de acierto en un 30% al emplear redes neuronales profundas con capas convolucionales.

Al igual que en la investigación conducida por Kaneshiro et Al. [21], se ha detectado en las matrices de confusión, una tendencia a clasificar con un mayor acierto las imágenes pertenecientes a la categoría de 'Cara humana'. Esto se debe a que los seres humanos estamos más acostumbrados a reconocer a nuestros semejantes, por lo que tiene sentido que se genere una respuesta más marcada y distintiva en nuestro cerebro al visualizar la cara de una persona.

Sin embargo, ha quedado patente que, con una base de datos de un tamaño tan reducido como la empleada, no se puede generalizar adecuadamente una red neuronal artificial para clasificar las lecturas de EEG. Además, se ha podido observar en figuras como la *Fig. 36*, que un mismo modelo puede generar respuestas muy variadas en función del sujeto. Esto se debe a la diferencia existente entre cerebros de personas distintas.

Por todo esto, para aplicar el aprendizaje profundo a un problema de este tipo, se debería contar con un mayor número de datos para su análisis, así como con un modelo de red, como el de validación cruzada, capaz de extraer características generales a varios sujetos, pudiendo generalizar los resultados obtenidos de esta forma.

4.1. Líneas de trabajo futuras. *Transfer learning* y los modelos pre-entrenados

Este proyecto se limita a abrir una puerta a la utilización del aprendizaje profundo para llevar a cabo investigaciones sobre la correlación de lo que un ser humano observa a su alrededor con lo que se refleja en su cerebro.
Para poder llevar esta investigación a un nivel más alto, se plantea la posibilidad de emplear modelos de redes neuronales ya pre-entrenados y realizar sobre ellos un proceso de *fine-tunning* para ajustarlos al problema planteado [44].
En la actualidad existen varios modelos pre entrenados que se caracterizan por su gran nivel de acierto en tareas de clasificación de imágenes. Estas redes han sido entrenadas con la base de datos Imagenet [45], que cuenta con más de 14 millones de imágenes y más de veinte mil categorías distintas, contando cada categoría con cientos de imágenes.
Los modelos más destacados tienen estructuras diferentes, pero todos ellos cuentan con tasas de acierto cercanas al 100%. Estos modelos son, por ejemplo, la red VGG en sus

versiones de 16 y 19 capas[46], una red más pequeña denominada MobileNet [47], o las redes ResNet [48][48] e Inception [49].

El tipo de imagen que se introduciría en estas redes no se corresponde con ninguna de las categorías que presentan como salida, por lo que habría que aplicar *transfer learning*. Este tipo de aprendizaje consiste en tomar una red preentrenada, suprimir sus últimas capas, generalmente densas, y sustituirlas por un modelo sencillo que entrenemos para ajustar los parámetros a las clases en las que se quiere clasificar la entrada.

BIBLIOGRAFÍA

[1] Sereno, M.I., Dale, A.M., Reppas, J.B., Kwong, K.K., Belliveau, J., Brady, T.J., Rosen, B.R. and Tootell, R.B.H., Borders of multiple visual areas in humans revealed by functional magnetic resonance imaging, *Science*, 268 (1995) 889–893.

[2] DiCarlo, J. J., Zoccolan, D., and Rust, N.C. "How does the brain solve visual object recognition?." *Neuron* 73.3 (2012): 415-434.

[3] Malach, R., et Al. Object-related activity revealed by functional magnetic resonance imaging in human occipital cortex. Proc Natl Acad Sci USA, 92 (1995), pp. 8135-8139

[4] N Kanwisher, J McDermott, MM Chun. The fusiform face area: a module in human extrastriate cortex specialized for face perception of outstanding interest J Neurosci, 17 (1997), pp. 4302-4311

[5] T.A Polk, M.J Farah. The neural development and organization of letter recognition: evidence from functional neuroimaging, computational modeling, and behavioral studies. Proc. Natl. Acad. Sci. USA, 95 (1998), pp. 847-852

[6] G.K. Aguirre, E. Zarahn, M. D'Esposito. An area within human ventral cortex sensitive to “building” stimuli: evidence and implications. Neuron, 21 (1998), pp. 373-383

[7] A. Ishai, L.G. Ungerleider, A. Martin, J.L. Schouten, J.V. HaxbyDistributed Representation of Objects in the Human Ventral Visual Pathway. Proc. Natl. Acad. Sci. USA, 96 (16) (1999), pp. 9379-9384

[8] Hanson, S. J., Matsuka, T., & Haxby, J. V. Combinatorial codes in ventral temporal lobe for object recognition: Haxby (2001) revisited: is there a “face” area?. *Neuroimage*, *23*.1 (2004): 156-166.

[9] Malmivuo J. Comparison of the properties of EEG and MEG in detecting the electric activity of the brain. Brain Topography. 2012;25(1):1–19. pmid:21912974

[10] Das K, Giesbrecht B, Eckstein MP. Predicting variations of perceptual performance across individuals from neural activity using pattern classifiers. NeuroImage. 2010;51(4): 1425–1437. Available from: http://www.sciencedirect.com/science/article/pii/S1053811910003174. pmid:20302949

[11] Simanova I, van Gerven M, Oostenveld R, Hagoort P. Identifying object categories from event-related EEG: Toward decoding of conceptual representations. PLoS ONE. 2010 12;5(12):e14465. Available from: http://dx.doi.org/10.1371%2Fjournal.pone.0014465. pmid:21209937

[12] Murphy B, Poesio M, Bovolo F, Bruzzone L, Dalponte M, Lakany H. EEG decoding of semantic category reveals distributed representations for single concepts. Brain and Language. 2011;117(1):12–22. Available from:

http://www.sciencedirect.com/science/article/pii/S0093934X10001811. pmid:21300399

[13] Shenoy P, Tan DS. Human-aided computing: Utilizing implicit human processing to classify images. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI'08. New York, NY, USA: ACM; 2008. p. 845–854. Available from: http://doi.acm.org/10.1145/1357054.1357188.

[14] van de Nieuwenhuijzen ME, Backus AR, Bahramisharif A, Doeller CF, Jensen O, van Gerven MAJ. MEG-based decoding of the spatiotemporal dynamics of visual category perception. NeuroImage. 2013;83(0):1063–1073. Available from: http://www.sciencedirect.com/science/article/pii/S1053811913008446. pmid:23927900

[15] Cauchoix M, Barragan-Jason G, Serre T, Barbeau EJ. The neural dynamics of face detection in the wild revealed by MVPA. The Journal of Neuroscience. 2014;34(3):846–854. Available from: http://www.jneurosci.org/content/34/3/846.abstract. pmid:24431443

[16] Tzovara A, Murray MM, Plomp G, Herzog MH, Michel CM, Lucia MD. Decoding stimulus-related information from single-trial EEG responses based on voltage topographies. Pattern Recognition. 2012;45(6):2109–2122. Brain Decoding. Available from: http://www.sciencedirect.com/science/article/pii/S0031320311001440.

[17] Clarke A, Devereux BJ, Randall B, Tyler LK. Predicting the time course of individual objects with MEG. Cerebral Cortex. 2014;Available from: http://cercor.oxfordjournals.org/content/early/2014/09/09/cercor.bhu203.abstract

[18] Kriegeskorte N, Mur M, Bandettini PA. Representational similarity analysis—connecting the branches of systems neuroscience. Frontiers in Systems Neuroscience. 2008;2(4). Available from: http://www.frontiersin.org/systems_neuroscience/10.3389/neuro.06.004.2008/abstract. pmid:19104670

[19] Kriegeskorte N, Mur M, Ruff DA, Kiani R, Bodurka J, Esteky H, et al. Matching categorical object representations in inferior temporal cortex of man and monkey. Neuron. 2008;60(6):1126–1141. Available from: http://www.sciencedirect.com/science/article/pii/S0896627308009434. pmid:19109916

[20] Cichy RM, Pantazis D, Oliva A. Resolving human object recognition in space and time. Nature neuroscience. 2014;17(3):455–462. pmid:24464044

[21] Kaneshiro, B., Guimaraes, M. P., Kim, H. S., Norcia, A. M., & Suppes, P. (2015). A representational similarity analysis of the dynamics of object processing using single-trial EEG classification. *Plos one*, *10*(8), e0135697.

[22] Ortega Asensio, E. (2017). Sistema de reconocimiento de gestos de la mano basado en procesamiento de imagen y redes neuronales convolucionales.

[23] Quintero-Rincon, Antonio & Liberczuk, Sergio & Risk, Marcelo. (2012). Preprocesamiento de EEG con Filtros Hampel. Argencon IEEE 2012. 89.

[24] Estupinan Donoso, A. Reducción de artefactos oculares en señales eeg: filtrado adaptativo como alternativa a la regresión lineal. 2009. <hal-00419918>

[25] Rodríguez Aldana Y, González Rubio T, Marañón Reyes E, Montoya Padrón A, Sanabria Macías F. Aplicación de corrección de artefactos en el electroencefalograma para monitoreo anestésico. Rev Cubana Neurol Neurocir. [Internet] 2015 [citado día, mes y año];5(Supl.1):S9–S14. Disponible en: http://www.revneuro.sld.cu/index.php/neu/article/view/250

[26] Biomedical Signal Acquisitions. Recuperado de: https://www.medicine.mcgill.ca/physio/vlab/biomed_signals/eeg_n.htm

[27] W. McCulloch and W. Pitts (1943). A Logical Calculus of ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics 5:115-133.

[28] Farobie, Obie & Hasanah, Nur & Matsumura, Yukihiko. (2015). Artificial Neural Network Modeling to Predict Biodiesel Production in Supercritical Methanol and Ethanol Using Spiral Reactor. Procedia Environmental Sciences. 28. 214-223. 10.1016/j.proenv.2015.07.028.

[29] Convolutional Neural Networks for Visual Recognition. Recuperado de: http://cs231n.github.io/neural-networks-1/

[30] Neural Networks and Back propagation explained in a simple way. Recuperado de: https://datathings.com/blog/post/neuralnet/

[31] Krizhevsky, A., Sutskever, I., & Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (2012). pp. 1097-1105.

[32] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

[33] Rawat, W., & Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, *29*(9) (2017) 2352-2449.

[34] Convolutional Neural Networks for Visual Recognition. Recuperado de: http://cs231n.github.io/convolutional-networks/

[35] A technical report on convolution arithmetic in the context of deep learning. Recuperado de: https://github.com/vdumoulin/conv_arithmetic

[36] Blair Kaneshiro, Marcos Perreau Guimaraes, Hyung-Suk Kim, Anthony M. Norcia, and Patrick Suppes (2015). EEG data analyzed in "A Representational Similarity Analysis of the Dynamics of Object Processing Using Single-Trial EEG Classification". Stanford Digital Repository. Available at: http://purl.stanford.edu/bq914sc3730

[37] Shuai, Lan & Gong, Tao. (2014). Temporal Relation between Top-down and Bottom-up Processing in Lexical Tone Perception. Frontiers in behavioral neuroscience. 8. 97. 10.3389/fnbeh.2014.00097.

[38] The Geodesic Sensor Net. Recuperado de: https://www.egi.com/research-division/geodesic-sensor-net

[39] A.J. Bell and T.J. Sejnowski. "An information maximization approach to blind separation and blind deconvolution," Neural Computation, 7(6), p.p. 1129-1159, 1995

[40] "Hola, Colaboratory". Recuperado de: https://colab.research.google.com/notebooks/welcome.ipynb

[41] "NVIDIA presenta el acelerador más rápido del mundo para análisis de datos y cálculo científico". Recuperado de: https://www.nvidia.es/object/tesla-k80-dual-gpu-accelerator-oct-14-2014-es.html

[42] "Tensorflow with GPU". Recuperado de: https://colab.research.google.com/notebooks/gpu.ipynb

[43] Wikimedia Commons contributors. File: K-fold cross validation.jpg [Internet]. Wikimedia Commons, the free media repository; 2016 Nov 17, 07:39 UTC. Recuperado de: https://commons.wikimedia.org/w/index.php?title=File:K-fold_cross_validation.jpg&oldid=214808292.

[44] Zhou, Z., Shin, J. Y., Zhang, L., Gurudu, S. R., Gotway, M. B., & Liang, J. (2017, July). Fine-Tuning Convolutional Neural Networks for Biomedical Image Analysis: Actively and Incrementally. In *CVPR* (pp. 4761-4772).

[45] Large Scale Visual Recognition Challenge. Recuperado de: http://www.image-net.org/challenges/LSVRC/

[46] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[47] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[48] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[49] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818-2826).

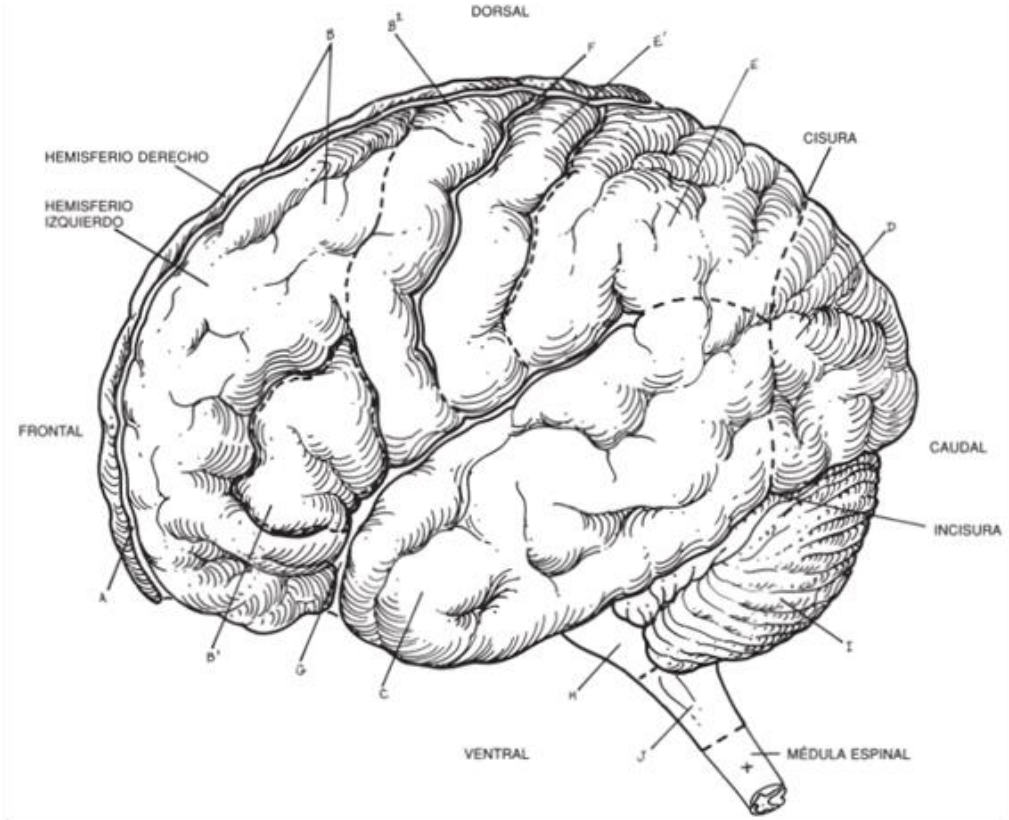
[50] Luque Ibañes, M. T. (2010). *Mejora de la herramienta MindReader: adquisición y análisis de señales EEG* (Master's thesis).

[51] Diamond, M. C., Scheibel, A. B., & Elson, L. M. (1996). El cerebro humano. Ariel Neurociencia.

[52] *Diccionario Enciclopédico Ilustrado de Medicina Dorland*. 1996. McGraw-Hill - Interamericana de España. Vol. 2. ISBN 84-7615-984-6

ANEXO A. MORFOLOGÍA DEL CEREBRO HUMANO

En este apartado se muestra una representación gráfica de las distintas partes de las que se compone el cerebro humano, con el fin de facilitar la comprensión de las partes mencionadas en el proyecto.



A – Cisura Longitudinal	B – Lóbulo Frontal	C – Lóbulo Temporal
D – Lóbulo Occipital	E – Lóbulo Parietal	F – Cisura Central
G – Cisura Lateral (Silvio)	H – Protuberancia	I – Cerebelo
J – Bulbo Raquídeo		

Fig. 39. Lóbulos y cisuras cerebrales [51].

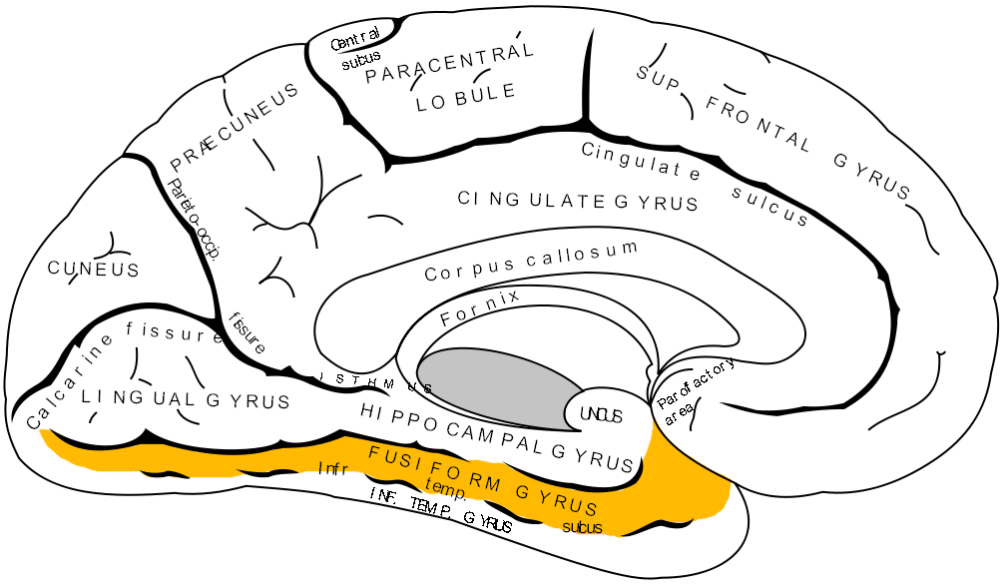


Fig. 38. Giros cerebrales [52].

ANEXO B. SOFTWARE DESARROLLADO

Este anexo recopila todo el código desarrollado durante este proyecto. Primero se presentarán dos códigos de preparación de la base de datos [36] en función de si se aplica a un solo sujeto o a todos ellos. A continuación, se adjuntarán los códigos correspondientes a los modelos propuestos en el epígrafe 2.4 de esta memoria.

Se debe tener en cuenta que el código se ha implementado empleando la herramienta online Google Colaboratory, por lo que todos los directorios que aparecen están referenciados a la nube, también de Google, Google Drive.

Todo el código se encuentra disponible también en la plataforma de libre acceso GitHub, a través del siguiente enlace: https://github.com/Angordil/Analisis-de-registros-de-EEG-mediante-redes-neuronales.git

Anexo B.1. Preparación de los datos para los 10 sujetos.

```
from scipy.io import loadmat

subjects_count = 10
subjects = []
for i in range(0, subjects_count):
    filename = "S" + str(i+1) + ".mat"
    subjects.append(loadmat("drive/TFG/data/" + filename))

subjects[0]


# Inspect the EEG data for the first trial of the first subject:

trial1 = subjects[0]['X_2D'][0]
print(trial1.shape)
print(trial1)

get_ipython().run_line_magic('matplotlib','inline')

import matplotlib.pyplot as plt

N = 32
plt.plot(trial1[:N])
plt.title("EEG Reading: Electrode 1, Trial 1, Subject 1")

import numpy as np

electrodes = 124
trial_image = np.reshape(trial1, (electrodes, N))
plt.imshow(trial_image, cmap = 'hot', interpolation = 'nearest')

# ### Data Preparation

# Concatenate all trials and output target labels from the first 9
test subjects

trials = np.concatenate([subjects[i]['X_2D'] for i in range(9)])
y_training = np.concatenate([subjects[i]['categoryLabels'][0] for i
in range(9)])

# Double-check number of trials equals number of target output labe
ls
```

```
print(len(trials),len(y_training))
print(y_training)

trials[0]

len(trials[0])

# ### Define Training and Validation/Holdout Sets

X_training = trials

holdout_fold = subjects[9]
X_validation = holdout_fold['X_2D']
print(len(X_validation), X_validation)

y_validation = holdout_fold['categoryLabels'][0]
print(len(y_validation),y_validation)
np.save('drive/TFG/validation/y_val', y_validation)

 ## 1-Hot Encode Target Output Labels
import tensorflow
from tensorflow import keras
from tensorflow.python.keras import utils


num_classes = 6
y_training1hot = utils.to_categorical(y_training - 1 , num_classes)
y_validation1hot = utils.to_categorical(y_validation -1, num_classe
s)

print y_training1hot.shape
print y_validation1hot.shape

np.save('drive/TFG/training/y_training1hot', y_training1hot)
np.save('drive/TFG/validation/y_validation1hot', y_validation1hot)

# Resize Training and Validation Sets

import numpy as np

X_training = np.reshape(X_training,(-1, electrodes, N,1))
X3_training = np.repeat(X_training,3, axis=3)
np.save('drive/TFG/training/X3_train',X3_training)

X_validation = np.reshape(X_validation, (-1, electrodes, N, 1))
X3_validation = np.repeat(X_validation, 3, axis=3)
np.save('drive/TFG/validation/X3_val', X3_validation)

print(X3_training.shape)
```

Anexo B.2. Preparación de los datos para un sujeto.

```
# Load data

from scipy.io import loadmat

subject = loadmat('drive/TFG/data/S1.mat')

# Inspect the EEG data for the first trial of the first subject:
X_training = subject['X_2D']
print(X_training.shape)

get_ipython().run_line_magic('matplotlib','inline')

import matplotlib.pyplot as plt

N = 32
plt.plot(X_training[:N])
plt.title("EEG Reading: Electrode 1, Trial 1, Subject 1")

import numpy as np

electrodes = 124
trial_image = np.reshape(X_training, (electrodes, N))
plt.imshow(trial_image, cmap = 'hot', interpolation = 'nearest')

# ### Data Preparation

y_training = subject['categoryLabels'][0]

# Double-check number of trials equals number of target output labels

print(len(X_training),len(y_training))
print(y_training)

print X_training.shape
len(X_training)

# Validation set will be defined when model.fit is done. It'll be a 10
% of the training data

## 1-Hot Encode Target Output Labels
import tensorflow
from tensorflow import keras
from tensorflow.python.keras import utils

num_classes = 6

y_training1hot = utils.to_categorical(y_training - 1 , num_classes)

print y_training1hot.shape

# Reshape training set

X_training = np.reshape(X_training,(-1, electrodes, N,1))
print X_training.shape

# Save set

np.save('drive/TFG/Trial only with subject 1/Datos/X_training', X_trai
ning)
np.save('drive/TFG/Trial only with subject 1/Datos/y_training1hot', y_
training1hot)
np.save('drive/TFG/Trial only with subject 1/Datos/y_training', y_trai
ning)
```

Anexo B.3. Código de obtención de gráficos y matriz de confusión

Este código se ha empleado para todos los modelos y permite obtener gráficos de precisión frente a tiempo, y error frente a tiempo, a partir del parámetro *history* de los modelos.

```
import matplotlib.pyplot as plt

## Print Accuracy Vs. Epochs
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('Accuracy over time')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show

## Print Loss Vs. Epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Cross Entropy Loss over Time')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show
# ### Model Quality Check: Confusion Matrix

import itertools
from sklearn.metrics import confusion_matrix

# Load validation set for one subject
y_validation = np.load('drive/TFG/Trial only with subject 1/Datos/y_tr
aining.npy')
y_validation = y_validation[-519:]
y_validation_predictions = modelk.predict(X_training[-519:], verbose =
1)

# Load validation set for all subjects
# y_validation = np.load('drive/TFG/validation/y_val.npy')
# y_validation_predictions = modelk.predict(X_validation, verbose = 1)

def plot_confusion_matrix(cm, classes, normalize = False, title = 'Con
fusion matrix', cmap = plt.cm.Blues):
  """
  This function prints and plots the confusion matrix.
  Normalization can be applied by setting 'normalize = True'
  """
  plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation = 45)
  plt.yticks(tick_marks, classes)

  if normalize:
    cm = cm.astype('float')/cm.sum(axis=1)[:,np.newaxis]
    print('Normalized confusion matrix')
  else:
    print('Confusion matrix, without normalization')

  print(cm)
```

```
  thresh = cm.max()/2.
  for i,j in itertools.product(range(cm.shape[0]), range(cm.shape[1]))
:
    plt.text(j, i, cm[i, j], horizontalalignment = 'center', color = '
white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cnf_matrix = confusion_matrix(y_validation - 1, np.argmax(y_validation
_predictions, axis = 1))

plt.figure()

class_names = ['Human Body', 'Human Face', 'Animal Body', 'Animal Face
', 'Fruit Vegetable', 'Inanimate Object']

plot_confusion_matrix(cnf_matrix, classes = class_names)
```

Anexo B.4. Modelo denso para los supuestos 1 y 3.

```
import numpy as np
import keras
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Dropout, Activati
on, Flatten
from tensorflow.python.keras.layers import Conv2D
from tensorflow.python.keras.applications import imagenet_utils

## Load training sets for one subject
X_training = np.load('drive/TFG/Trial only with subject 1/Datos/X_t
raining.npy')
y_training1hot = np.load('drive/TFG/Trial only with subject 1/Datos
/y_training1hot.npy')

## Load training and validaton sets for all subjects
# X_training = np.load('drive/TFG/training/X3_train.npy')
# y_training1hot = np.load('drive/TFG/training/y_training1hot.npy')
# X_validation = np.load('drive/TFG/validation/X3_val.npy')
# y_validation1hot = np.load('drive/TFG/validation/y_validation1hot
.npy')

print X_training.shape
print y_training1hot.shape

num_classes = 6

model = Sequential()

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

# Train model for one subject
history = model.fit(X_training, y_training1hot, epochs = 24, valida
tion_split = 0.1, shuffle = True)

# Train model for all subjects
# history = model.fit(X_training, y_training1hot, epochs = 24, vali
dation_data = [X_validation, y_validation1hot], shuffle = True)
```

Anexo B.5. Modelo convolucional simple para los supuestos 1 y 3.

```
import numpy as np
import keras
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Dropout, Activati
on, Flatten, MaxPooling2D
from tensorflow.python.keras.layers import Conv2D
from tensorflow.python.keras.applications import imagenet_utils

## Load training sets for one subject
X_training = np.load('drive/TFG/Trial only with subject 1/Datos/X_t
raining.npy')
y_training1hot = np.load('drive/TFG/Trial only with subject 1/Datos
/y_training1hot.npy')

## Load training and validation sets for all subjects

# X_training = np.load('drive/TFG/training/X3_train.npy')
# y_training1hot = np.load('drive/TFG/training/y_training1hot.npy')
# X_validation = np.load('drive/TFG/validation/X3_val.npy')
# y_validation1hot = np.load('drive/TFG/validation/y_validation1hot
.npy')

print X_training.shape
print y_training1hot.shape

num_classes = 6

model = Sequential()

model.add(Conv2D(16, (3,3), input_shape = (124,32,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D())
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

## Train model for one subject
history = model.fit(X_training, y_training1hot, epochs = 48, valida
tion_split = 0.1, shuffle = True)

## Train model for all subjects
# history = model.fit(X_training, y_training1hot, epochs = 48, vali
dation_data = [X_validation, y_validation1hot], shuffle = True)
```

Anexo B.6. Modelo convolucional doble para los supuestos 1 y 3.

```
import numpy as np
import keras
from tensorflow.python.keras.models import Sequential, save_model
from tensorflow.python.keras.layers import Dense, Dropout, Activation,
Flatten
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D, Avera
gePooling2D
from tensorflow.python.keras.applications import imagenet_utils
import tensorflow

## Load training sets for one subject
X_training = np.load('drive/TFG/Trial only with subject 1/Datos/X_trai
ning.npy')
y_training1hot = np.load('drive/TFG/Trial only with subject 1/Datos/y_
training1hot.npy')

## Load training and validation sets for all subjects
# X_training = np.load('drive/TFG/training/X3_train.npy')
# y_training1hot = np.load('drive/TFG/training/y_training1hot.npy')
# X_validation = np.load('drive/TFG/validation/X3_val.npy')
# y_validation1hot = np.load('drive/TFG/validation/y_validation1hot.np
y')

print X_training.shape
print y_training1hot.shape

num_classes = 6

model = Sequential()

model.add(Conv2D(16,(3,3), input_shape = (124,32,1)))
# model.add(Conv2D(64, (3,3), input_shape = (124,64,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D())

model.add(Conv2D(128,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D())
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', m
etrics = ['accuracy'])

## Train model for one subject
history = model.fit(X_training, y_training1hot, epochs = 24, validatio
n_split = 0.1, shuffle = True, verbose = 1)

## Train model for all subjects
# history = model.fit(X_training, y_training1hot, epochs = 24, validat
ion_data = [X_validation, y_validation1hot], shuffle = True, verbose =
1)
```

Anexo B.7. Modelo de votación.

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow
from tensorflow import keras
from tensorflow.python.keras import utils
from tensorflow.python.keras.models import Sequential, save_model, loa
d_model
from tensorflow.python.keras.layers import Dense, Dropout, Activation,
Flatten
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D
from tensorflow.python.keras.applications import imagenet_utils
from scipy.io import loadmat
import itertools
from sklearn.metrics import confusion_matrix

def doubleconv(X_training, y_training1hot, subj_num):

  num_classes = 6

  model = Sequential()

  model.add(Conv2D(64, (3,3), input_shape = (124,32,1)))
  model.add(Activation('relu'))
  model.add(MaxPooling2D())

  model.add(Conv2D(128,(3,3)))
  model.add(Activation('relu'))
  model.add(MaxPooling2D())
  model.add(Dropout(0.3))

  model.add(Flatten())
  model.add(Dense(256))
  model.add(Activation('relu'))
  model.add(Dropout(0.3))
  model.add(Dense(64))
  model.add(Activation('relu'))
  model.add(Dropout(0.3))
  model.add(Dense(num_classes))
  model.add(Activation('softmax'))

  model.compile(loss='categorical_crossentropy', optimizer = 'adam', m
etrics = ['accuracy'])

    # Train model

  history = model.fit(X_training, y_training1hot, epochs = 24, validat
ion_split = 0.1, shuffle = True, verbose = 1)

  #model.save_weights('drive/TFG/Trial only with subject 1/Weights/wei
ghtsVoteModel'+str(subj_num+1)+'.hdf5')

  return model, history

def data_preparation(subj_num):
  # Load data

  subject = loadmat('drive/TFG/data/S'+str(subj_num+1)+'.mat')

  # Inspect the EEG data for the first trial of the first subject:

  X_training = subject['X_2D']

  N = 32
```

```
  electrodes = 124

  # ### Data Preparation

  y_training = subject['categoryLabels'][0]

  # Validation set will be defined when model.fit is done. It'll be a
10% of the training data

  ## 1-Hot Encode Target Output Labels
  num_classes = 6

  y_training1hot = utils.to_categorical(y_training - 1 , num_classes)

  print y_training1hot.shape

  # Reshape training set

  X_training = np.reshape(X_training,(-1, electrodes, N,1))

  return X_training, y_training1hot, y_training


for subj_num in range(0,9):
  X_training, y_training1hot, y_training = data_preparation(subj_num)
  model, history = doubleconv(X_training,y_training1hot, subj_num)
  #conf_matrix(y_training, X_training, subj_num, model)
  save_model(model, 'drive/TFG/Trial only with subject 1/model'+str(su
bj_num+1)+'.h5py')
  print(str(subj_num) + ' - ok')
# Load models
model1 = load_model('drive/TFG/Trial only with subject 1/model1.h5py')
model2 = load_model('drive/TFG/Trial only with subject 1/model2.h5py')
model3 = load_model('drive/TFG/Trial only with subject 1/model3.h5py')
model4 = load_model('drive/TFG/Trial only with subject 1/model4.h5py')
model5 = load_model('drive/TFG/Trial only with subject 1/model5.h5py')
model6 = load_model('drive/TFG/Trial only with subject 1/model6.h5py')
model7 = load_model('drive/TFG/Trial only with subject 1/model7.h5py')
model8 = load_model('drive/TFG/Trial only with subject 1/model8.h5py')
model9 = load_model('drive/TFG/Trial only with subject 1/model9.h5py')
X_validation,y_validation1hot, y_validation = data_preparation(9)

trial = X_validation[:,:,:,:]

print trial.shape

## Predict class
val1 = np.expand_dims(np.argmax(model1.predict(trial, verbose = 0), ax
is = 1),axis = 1)
val2 = np.expand_dims(np.argmax(model2.predict(trial, verbose = 0), ax
is = 1),axis = 1)
val3 = np.expand_dims(np.argmax(model3.predict(trial, verbose = 0), ax
is = 1),axis = 1)
val4 = np.expand_dims(np.argmax(model4.predict(trial, verbose = 0), ax
is = 1),axis = 1)
val5 = np.expand_dims(np.argmax(model5.predict(trial, verbose = 0), ax
is = 1),axis = 1)
val6 = np.expand_dims(np.argmax(model6.predict(trial, verbose = 0), ax
is = 1),axis = 1)
val7 = np.expand_dims(np.argmax(model7.predict(trial, verbose = 0), ax
is = 1),axis = 1)
val8 = np.expand_dims(np.argmax(model8.predict(trial, verbose = 0), ax
is = 1),axis = 1)
val9 = np.expand_dims(np.argmax(model9.predict(trial, verbose = 0), ax
is = 1),axis = 1)
```

```
val = np.concatenate((val1,val2,val3,val4,val5,val6,val7,val8,val9),ax
is = 1)
print val.shape

from scipy import stats

predicted_classes = stats.mode(val, axis =1)[0]+1
print predicted_classes.shape

## Compare predicted classes with actual classes
correct = 0
for i in range(0, len(y_validation)):
  if(y_validation[i]==predicted_classes[i]):
    correct = correct +1
  else: correct = correct

accuracy = correct*100/len(y_validation)
print(str(accuracy)+ ' %')
```

Anexo B.8. Validación cruzada y simple para el modelo denso en el supuesto 2.

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow
from tensorflow import keras
from tensorflow.python.keras import utils
from tensorflow.python.keras.models import Sequential, save_model, loa
d_model
from tensorflow.python.keras.layers import Dense, Dropout, Activation,
Flatten
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D
from tensorflow.python.keras.applications import imagenet_utils
from scipy.io import loadmat
import itertools
from sklearn.metrics import confusion_matrix

def dense(X_training, y_training1hot, X_validation, y_validation1hot):

  num_classes = 6

  model = Sequential()

  model.add(Flatten())
  model.add(Dense(128))
  model.add(Activation('relu'))
  model.add(Dropout(0.3))
  model.add(Dense(64))
  model.add(Activation('relu'))
  model.add(Dropout(0.3))
  model.add(Dense(num_classes))
  model.add(Activation('softmax'))

  model.compile(loss='categorical_crossentropy', optimizer = 'adam', m
etrics = ['accuracy'])

    # Train model

  history = model.fit(X_training, y_training1hot, epochs = 48, validat
ion_data = (X_validation, y_validation1hot), shuffle = True, verbose =
1)

  return model, history

def data_preparation(subj_num):
  # Load data

  subject = loadmat('drive/TFG/data/S'+str(subj_num+1)+'.mat')

  # Inspect the EEG data for the first trial of the first subject:
  X_training = subject['X_2D']

  N = 32
  electrodes = 124

  # ### Data Preparation

  y_training = subject['categoryLabels'][0]

  # Validation set will be defined when model.fit is done. It'll be a
10% of the training data

  ## 1-Hot Encode Target Output Labels
  y_training1hot = utils.to_categorical(y_training - 1 , num_classes)
```

```
  print y_training1hot.shape

  # Reshape training set

  X_training = np.reshape(X_training,(-1, electrodes, N,1))

  return X_training, y_training1hot, y_training

# Data preparation and concatenation
X_train, y_train1hot, y_train = data_preparation(0)

for subj in range(1,10):
  X_training, y_training1hot, y_training = data_preparation(subj)
  X_train = np.append(X_train, X_training, axis = 0)
  y_train1hot = np.append(y_train1hot, y_training1hot, axis = 0)
  y_train = np.append(y_train, y_training, axis = 0)
  print('Subj ' + str(subj) + ' added')

random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))

# Simple validation --> one loops, 10% as validation
random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))
start = 0

fin = start+5186
if fin > len(X_train):
  fin = len(X_train)

Xval = X_train[random_index_training[start:fin]]
yval1hot = y_train1hot[random_index_training[start:fin]]
yval = y_train[random_index_training[start:fin]]
if start == 0:
  Xtrain = X_train[random_index_training[fin:]]
  ytrain1hot = y_train1hot[random_index_training[fin:]]
else:
  Xtrain = X_train[random_index_training[0:start]]
  x = X_train[random_index_training[fin:]]
  Xtrain = np.append(Xtrain, x, axis = 0)

  ytrain1hot = y_train1hot[random_index_training[0:start]]
  y = y_train1hot[random_index_training[fin:]]
  ytrain1hot = np.append(ytrain1hot, y, axis = 0)

model, history = dense(Xtrain, ytrain1hot, Xval, yval1hot)
prediction = model.predict(Xval, verbose = 0)
val = np.argmax(prediction, axis = 1)


## Compare predicted classes with actual classes
correct = 0
for i in range(0, len(yval)):
  if(yval[i]==(val[i]+1)):
    correct = correct +1
  else: correct = correct
accuracy = correct*100/len(yval)
print (str(accuracy) + ' % - loop ')

start = fin

## Print confusion matrix
conf_matrix(yval, Xval, model)
```

```
### Cross validation --> ten loops, 10% as validation
random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))
start = 0

for loop in range(0,10):
  fin = start+5186
  if fin > len(X_train):
    fin = len(X_train)

  Xval = X_train[random_index_training[start:fin]]
  yval1hot = y_train1hot[random_index_training[start:fin]]
  yval = y_train[random_index_training[start:fin]]
  if start == 0:
    Xtrain = X_train[random_index_training[fin:]]
    ytrain1hot = y_train1hot[random_index_training[fin:]]
  else:
    Xtrain = X_train[random_index_training[0:start]]
    x = X_train[random_index_training[fin:]]
    Xtrain = np.append(Xtrain, x, axis = 0)

    ytrain1hot = y_train1hot[random_index_training[0:start]]
    y = y_train1hot[random_index_training[fin:]]
    ytrain1hot = np.append(ytrain1hot, y, axis = 0)

  model, history = dense(Xtrain, ytrain1hot, Xval, yval1hot)
  prediction = model.predict(Xval, verbose = 0)
  val = np.argmax(prediction, axis = 1)


  ## Compare predicted classes with actual classes
  correct = 0
  for i in range(0, len(yval)):
    if(yval[i]==(val[i]+1)):
      correct = correct +1
    else: correct = correct

  accuracy = correct*100/len(yval)
  print (str(accuracy) + ' % - loop ' + str(loop+1))

  if loop == 0:
    Acc = [accuracy]
  else:
    Acc = np.append(Acc, [accuracy], axis = 0)
  start = fin

  ## Print confusion matrix
  conf_matrix(yval, Xval, model)

print("Final accuracy: " + str(Acc) + "%")
```

Anexo B.9. Validación cruzada y simple para el modelo convolucional simple en el supuesto 2.

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow
from tensorflow import keras
from tensorflow.python.keras import utils
from tensorflow.python.keras.models import Sequential, save_model, loa
d_model
from tensorflow.python.keras.layers import Dense, Dropout, Activation,
Flatten
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D
from tensorflow.python.keras.applications import imagenet_utils
from scipy.io import loadmat
import itertools
from sklearn.metrics import confusion_matrix


def conv(X_training, y_training1hot, X_validation, y_validation1hot):

  num_classes = 6

  model = Sequential()

  model.add(Conv2D(16, (3,3), input_shape = (124,32,1)))
  model.add(Activation('relu'))
  model.add(MaxPooling2D())
  model.add(Dropout(0.3))

  model.add(Flatten())
  model.add(Dense(128))
  model.add(Activation('relu'))
  model.add(Dropout(0.3))
  model.add(Dense(64))
  model.add(Activation('relu'))
  model.add(Dropout(0.3))
  model.add(Dense(num_classes))
  model.add(Activation('softmax'))

  model.compile(loss='categorical_crossentropy', optimizer = 'adam', m
etrics = ['accuracy'])

    # Train model

  history = model.fit(X_training, y_training1hot, epochs = 84, validat
ion_data = (X_validation, y_validation1hot), shuffle = True, verbose =
1)

  #model.save_weights('drive/TFG/Trial only with subject 1/Weights/wei
ghtsVoteModel'+str(subj_num+1)+'.hdf5')

  return model, history

def data_preparation(subj_num):
  # Load data

  subject = loadmat('drive/TFG/data/S'+str(subj_num+1)+'.mat')

  # Inspect the EEG data for the first trial of the first subject:
  X_training = subject['X_2D']

  N = 32
  electrodes = 124
```

```
  # ### Data Preparation

  y_training = subject['categoryLabels'][0]

  # Validation set will be defined when model.fit is done. It'll be a
10% of the training data

  ## 1-Hot Encode Target Output Labels
  num_classes = 6

  y_training1hot = utils.to_categorical(y_training - 1 , num_classes)

  print y_training1hot.shape

  # Reshape training set

  X_training = np.reshape(X_training,(-1, electrodes, N,1))

  return X_training, y_training1hot, y_training

### Data preparation and concatenation
X_train, y_train1hot, y_train = data_preparation(0)

for subj in range(1,10):
  X_training, y_training1hot, y_training = data_preparation(subj)
  X_train = np.append(X_train, X_training, axis = 0)
  y_train1hot = np.append(y_train1hot, y_training1hot, axis = 0)
  y_train = np.append(y_train, y_training, axis = 0)
  print('Subj ' + str(subj) + ' added')

random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))


### Simple validation --> one loops, 10% as validation
random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))
start = 0

fin = start+5186
if fin > len(X_train):
  fin = len(X_train)

Xval = X_train[random_index_training[start:fin]]
yval1hot = y_train1hot[random_index_training[start:fin]]
yval = y_train[random_index_training[start:fin]]
if start == 0:
  Xtrain = X_train[random_index_training[fin:]]
  ytrain1hot = y_train1hot[random_index_training[fin:]]
else:
  Xtrain = X_train[random_index_training[0:start]]
  x = X_train[random_index_training[fin:]]
  Xtrain = np.append(Xtrain, x, axis = 0)

  ytrain1hot = y_train1hot[random_index_training[0:start]]
  y = y_train1hot[random_index_training[fin:]]
  ytrain1hot = np.append(ytrain1hot, y, axis = 0)

model, history = conv(Xtrain, ytrain1hot, Xval, yval1hot)
prediction = model.predict(Xval, verbose = 0)
val = np.argmax(prediction, axis = 1)


## Compare predicted classes with actual classes
```

```
correct = 0
for i in range(0, len(yval)):
  if(yval[i]==(val[i]+1)):
    correct = correct +1
  else: correct = correct

accuracy = correct*100/len(yval)
print (str(accuracy) + ' % - loop ' )

start = fin

conf_matrix(yval, Xval, model)


### Cross validation --> ten loops, 10% as validation
random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))
start = 0

for loop in range(0,10):
  fin = start+5186
  if fin > len(X_train):
    fin = len(X_train)

  Xval = X_train[random_index_training[start:fin]]
  yval1hot = y_train1hot[random_index_training[start:fin]]
  yval = y_train[random_index_training[start:fin]]
  if start == 0:
    Xtrain = X_train[random_index_training[fin:]]
    ytrain1hot = y_train1hot[random_index_training[fin:]]
  else:
    Xtrain = X_train[random_index_training[0:start]]
    x = X_train[random_index_training[fin:]]
    Xtrain = np.append(Xtrain, x, axis = 0)

    ytrain1hot = y_train1hot[random_index_training[0:start]]
    y = y_train1hot[random_index_training[fin:]]
    ytrain1hot = np.append(ytrain1hot, y, axis = 0)

  model, history = conv(Xtrain, ytrain1hot, Xval, yval1hot)
  prediction = model.predict(Xval, verbose = 0)
  val = np.argmax(prediction, axis = 1)


  ## Compare predicted classes with actual classes
  correct = 0
  for i in range(0, len(yval)):
    if(yval[i]==(val[i]+1)):
      correct = correct +1
    else: correct = correct

  accuracy = correct*100/len(yval)
  print (str(accuracy) + ' % - loop ' + str(loop+1))
  if loop == 0:
    Acc = [accuracy]
  else:
    Acc = np.append(Acc, [accuracy], axis = 0)

  start = fin

  conf_matrix(yval, Xval, model)

print("Final accuracy: " + str(Acc) + "%")
```

Anexo B.10. Validación cruzada y simple para el modelo convolucional doble en el supuesto 2.

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow
from tensorflow import keras
from tensorflow.python.keras import utils
from tensorflow.python.keras.models import Sequential, save_model, loa
d_model
from tensorflow.python.keras.layers import Dense, Dropout, Activation,
Flatten
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D
from tensorflow.python.keras.applications import imagenet_utils
from scipy.io import loadmat
import itertools
from sklearn.metrics import confusion_matrix


def doubleconv(X_training, y_training1hot, X_validation, y_validation1
hot):

  num_classes = 6

  model = Sequential()

  model.add(Conv2D(16, (3,3), input_shape = (124,32,1)))
  model.add(Activation('relu'))
  model.add(MaxPooling2D())

  model.add(Conv2D(128,(3,3)))
  model.add(Activation('relu'))
  model.add(MaxPooling2D())
  model.add(Dropout(0.3))

  model.add(Flatten())
  model.add(Dense(256))
  model.add(Activation('relu'))
  model.add(Dropout(0.3))
  model.add(Dense(64))
  model.add(Activation('relu'))
  model.add(Dropout(0.3))
  model.add(Dense(num_classes))
  model.add(Activation('softmax'))

  model.compile(loss='categorical_crossentropy', optimizer = 'adam', m
etrics = ['accuracy'])

    # Train model

  history = model.fit(X_training, y_training1hot, epochs = 64, validat
ion_data = (X_validation, y_validation1hot), shuffle = True, verbose =
1)

  return model

def data_preparation(subj_num):
  # Load data

  subject = loadmat('drive/TFG/data/S'+str(subj_num+1)+'.mat')

  # Inspect the EEG data for the first trial of the first subject:

  X_training = subject['X_2D']
```

```
  N = 32
  electrodes = 124

  # ### Data Preparation

  y_training = subject['categoryLabels'][0]

  # Validation set will be defined when model.fit is done. It'll be a
10% of the training data

  ## 1-Hot Encode Target Output Labels
  num_classes = 6

  y_training1hot = utils.to_categorical(y_training - 1 , num_classes)

  print y_training1hot.shape

  # Reshape training set

  X_training = np.reshape(X_training,(-1, electrodes, N,1))

  return X_training, y_training1hot, y_training


### Data preparation and concatenation
X_train, y_train1hot, y_train = data_preparation(0)

for subj in range(1,10):
  X_training, y_training1hot, y_training = data_preparation(subj)
  X_train = np.append(X_train, X_training, axis = 0)
  y_train1hot = np.append(y_train1hot, y_training1hot, axis = 0)
  y_train = np.append(y_train, y_training, axis = 0)
  print('Subj ' + str(subj) + ' added')

random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))


### Simple validation --> one loops, 10% as validation
random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))
start = 0

fin = start+5186

Xval = X_train[random_index_training[start:fin]]
yval1hot = y_train1hot[random_index_training[start:fin]]
yval = y_train[random_index_training[start:fin]]
if start == 0:
  Xtrain = X_train[random_index_training[fin:]]
  ytrain1hot = y_train1hot[random_index_training[fin:]]
else:
  Xtrain = X_train[random_index_training[0:start]]
  x = X_train[random_index_training[fin:]]
  Xtrain = np.append(Xtrain, x, axis = 0)

  ytrain1hot = y_train1hot[random_index_training[0:start]]
  y = y_train1hot[random_index_training[fin:]]
  ytrain1hot = np.append(ytrain1hot, y, axis = 0)

model, history = doubleconv(Xtrain, ytrain1hot, Xval, yval1hot)
prediction = model.predict(Xval, verbose = 0)
val = np.argmax(prediction, axis = 1)
```

```
## Compare predicted classes with actual classes
correct = 0
for i in range(0, len(yval)):
  if(yval[i]==(val[i]+1)):
    correct = correct +1
  else: correct = correct
accuracy = correct*100/len(yval)
print (str(accuracy) + ' % - loop ')

start = fin

conf_matrix(yval, Xval, model)


### Cross validation --> ten loops, 10% as validation
random_index_training = np.random.randint(len(X_train), size = len(X_t
rain))
start = 0

for loop in range(0,10):
  fin = start+5186
  if fin > len(X_train):
    fin = len(X_train)

  Xval = X_train[random_index_training[start:fin]]
  yval1hot = y_train1hot[random_index_training[start:fin]]
  yval = y_train[random_index_training[start:fin]]
  if start == 0:
    Xtrain = X_train[random_index_training[fin:]]
    ytrain1hot = y_train1hot[random_index_training[fin:]]
  else:
    Xtrain = X_train[random_index_training[0:start]]
    x = X_train[random_index_training[fin:]]
    Xtrain = np.append(Xtrain, x, axis = 0)

    ytrain1hot = y_train1hot[random_index_training[0:start]]
    y = y_train1hot[random_index_training[fin:]]
    ytrain1hot = np.append(ytrain1hot, y, axis = 0)

  model = doubleconv(Xtrain, ytrain1hot, Xval, yval1hot)
  prediction = model.predict(Xval, verbose = 0)
  val = np.argmax(prediction, axis = 1)

  ## Compare predicted classes with actual classes
  correct = 0
  for i in range(0, len(yval)):
    if(yval[i]==(val[i]+1)):
      correct = correct +1
    else: correct = correct

  accuracy = correct*100/len(yval)
  print (str(accuracy) + ' % - loop ' + str(loop+1))
  if loop == 0:
    Acc = [accuracy]
  else:
    Acc = np.append(Acc, [accuracy], axis = 0)

  start = fin

  conf_matrix(yval, Xval, model)

print("Final accuracy: " + str(Acc) + "%")
```